BPM v7.5 – Straight Through Processing in Integration Designer

Lab Exercise

IBM BPM 7.5 Proof of Technology

Lab Number 8

© Copyright IBM Corporation, 2011

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

LAB 8 BUILDING A STRAIGHT THROUGH PROCESS IN INTEGRATION DESIGNER 7
- 1.1 OVERVIEW 7
- 1.2 START UP 7
  - 1.2.1 STARTING THE PROCESS CENTER 7
  - 1.2.2 STARTING IBM INTEGRATION DESIGNER (IID) 8
- 1.3 POPULATING THE LIBRARY MODULE 11
  - 1.3.1 CREATE BUSINESS OBJECTS 13
  - 1.3.2 CREATE INTERFACES 21
- 1.4 CREATING THE BUSINESS PROCESS 29
  - 1.4.1 CREATE A BPEL PROCESS 29
  - 1.4.2 ADD REFERENCE PARTNERS 31
  - 1.4.3 ADD VARIABLES 32
  - CONSTRUCTING THE BPEL PROCESS 33
  - 1.4.4 ADD A FAULT HANDLER TO THE BPEL PROCESS 38
  - 1.4.5 COMPLETED BPEL PROCESS – ORDER PROCESSING 41
- 1.5 ASSEMBLE THE ORDER PROCESSING MODULE 42
  - 1.5.1 ADD THE ORDER PROCESSING PROCESS 42
  - 1.5.2 ADD JAVA COMPONENTS 42
  - 1.5.3 WIRE THE ASSEMBLY DIAGRAM 50
- 1.6 TESTING THE PROCESS 51
  - 1.6.1 POSITIVE RESULTS TEST 51
  - 1.6.2 NEGATIVE RESULTS TEST 55
- 1.7 SUMMARY 56

APPENDIX – CREATING A TEST SERVER ENVIRONMENT 57

Lab 8 Building a Straight Through Process in Integration Designer

This document describes the steps to create a simple Straight Through Process (STP) application in Business Process Manager v7.5 (BPM). A STP is a business process with no human intervention and in this example it will be a short running process or micro-flow depicting an order processing system. The process will be created and tested using the IBM Integration Designer V7.5 (IID).

1.1 Overview

The scenario of this lab is based on the order process, in which Internet customers submit an order on a Web page. During the processing of the order, a shipment service is contacted from the shipment price. A charging service is also called, that processes the billing of the customer.

1.2 Start Up

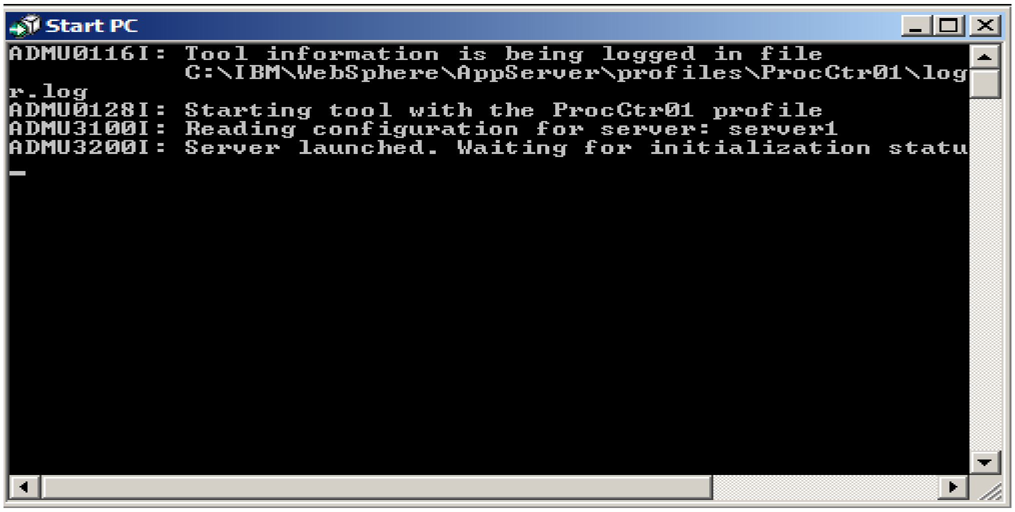
In this section, we will assume that the IBM Process Center (PC) has not been started. We will show you how to start both the Process Center and IBM Integration Designer V7.5. These are the two tools that you will be using throughout this lab.

NOTE: Process Center might already be running. You can skip to Section 1.2.2.

1.2.1 Starting the Process Center

__1. In order to start the **Process Center** double-click the icon on the desktop.

__2. The following window will appear and stay open while the Process Center server starts.



```
ADMU0116I: Tool information is being logged in file
           C:\IBM\WebSphere\AppServer\profiles\ProcCtr01\log
r.log
ADMU0128I: Starting tool with the ProcCtr01 profile
ADMU3100I: Reading configuration for server: server1
ADMU3200I: Server launched. Waiting for initialization statu
_
```

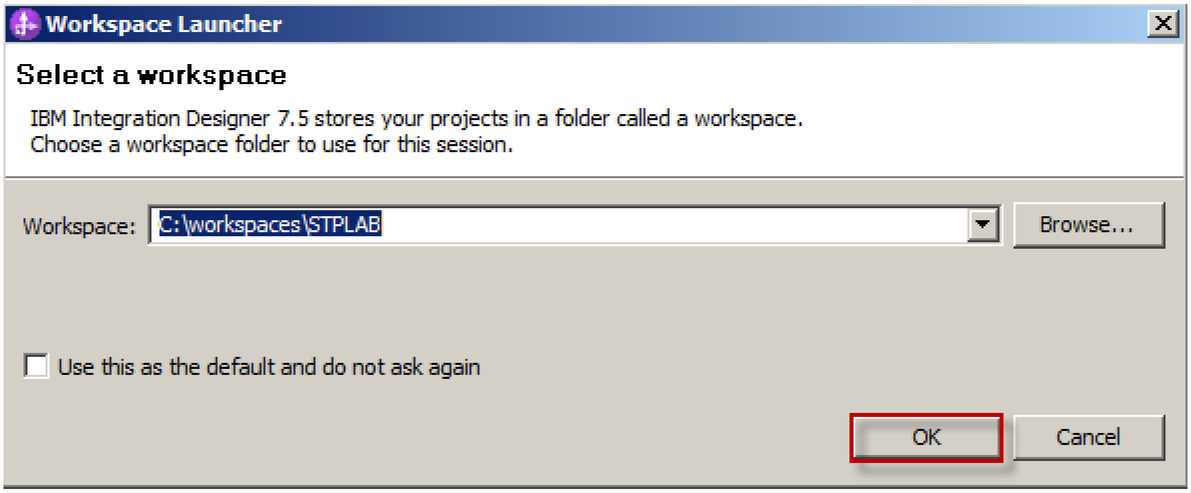
__3. The window will automatically close when the server has been started.

1.2.2 Starting IBM Integration Designer (IID)

__1. Start IID by either via the Start Programs menu and choosing

or find it on the desktop.

- __a. IID will start and ask where to store the workspace artifacts. IID has two modes of operation, online connected to the Process Center and offline all components stored locally.
- __**b.** For the workspace enter **C:\workspaces\STPLAB**
- __**c.** Click the **OK** button



Select a workspace

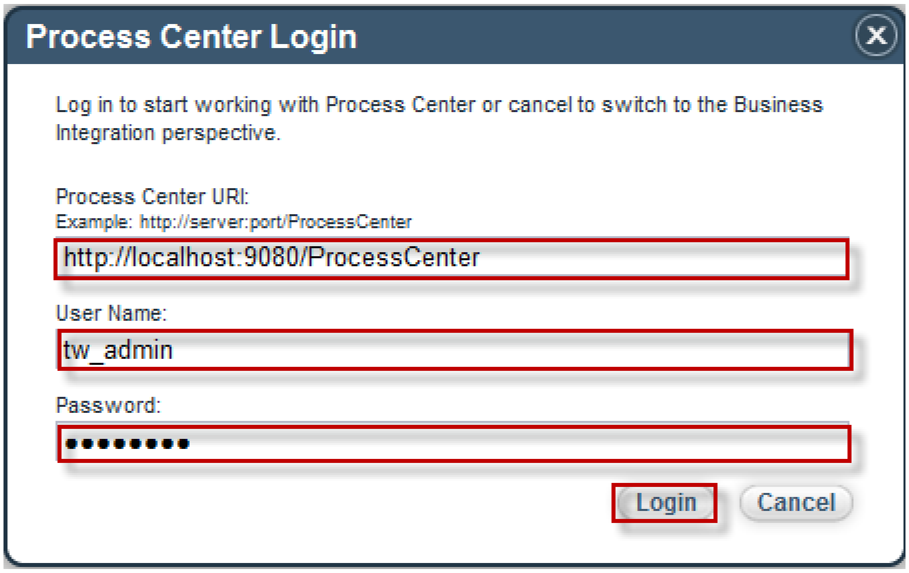
IBM Integration Designer 7.5 stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\workspaces\STPLAB

☐ Use this as the default and do not ask again

__2. The **Process Center Login** dialog will pop-up. If you wanted to work off-line you would use the Cancel button.

- __a. For the **Process Center URL**, enter http://localhost:9080/ProcessCenter**.**
- __b. Enter **tw_admin** for the **User Name and Password**.
- __c. Click the **Login** button



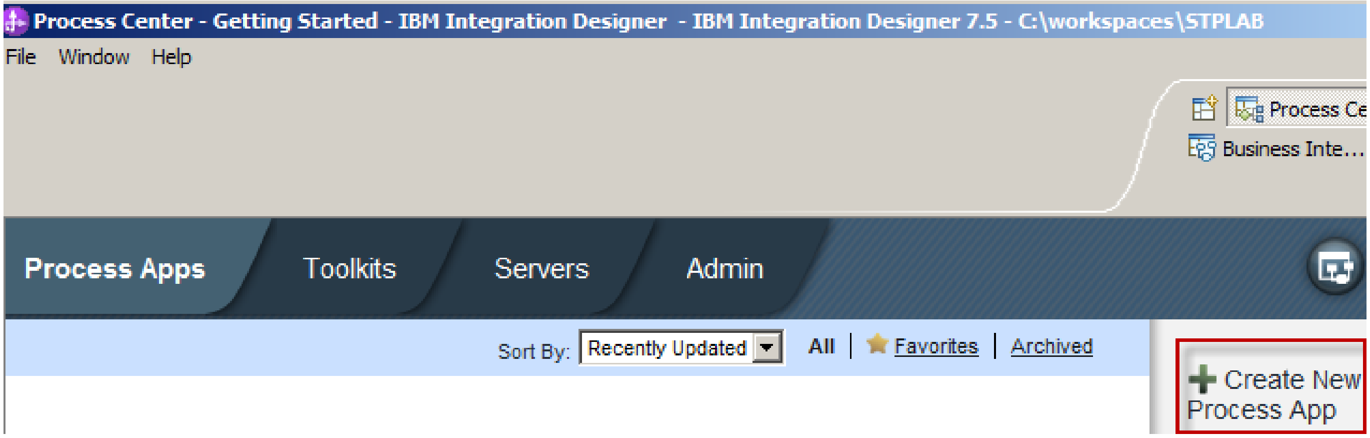
Log in to start working with Process Center or cancel to switch to the Business Integration perspective.

Process Center URI:
Example: http://server:port/ProcessCenter
http://localhost:9080/ProcessCenter

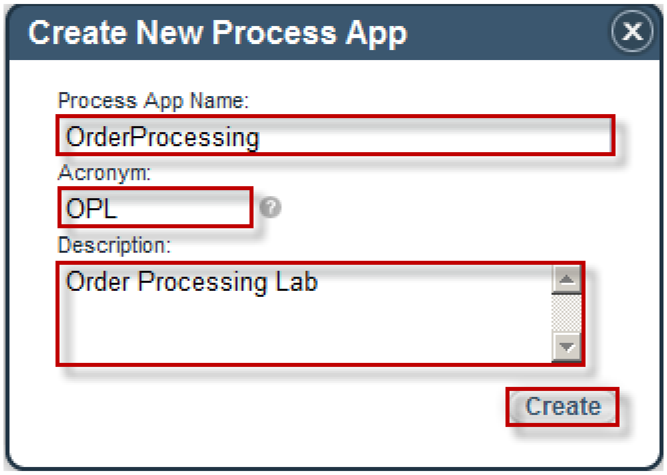
User Name:
tw_admin

Password:
●●●●●●●●●

__3. If the Getting Started screen shows up, close it.
  __a. Make sure you are in the **Process Center** perspective.


Process Center - Getting Started - IBM Integration Designer - IBM Integration Designer 7.5 - C:\workspaces\STPLAB
File Window Help
Process Ce
Business Inte…
Process Apps
Toolkits
Servers
Admin
Sort By: Recently Updated
All | Favorites | Archived
Create New Process App

__4. Create a new process application by clicking the **Create New Process Application**



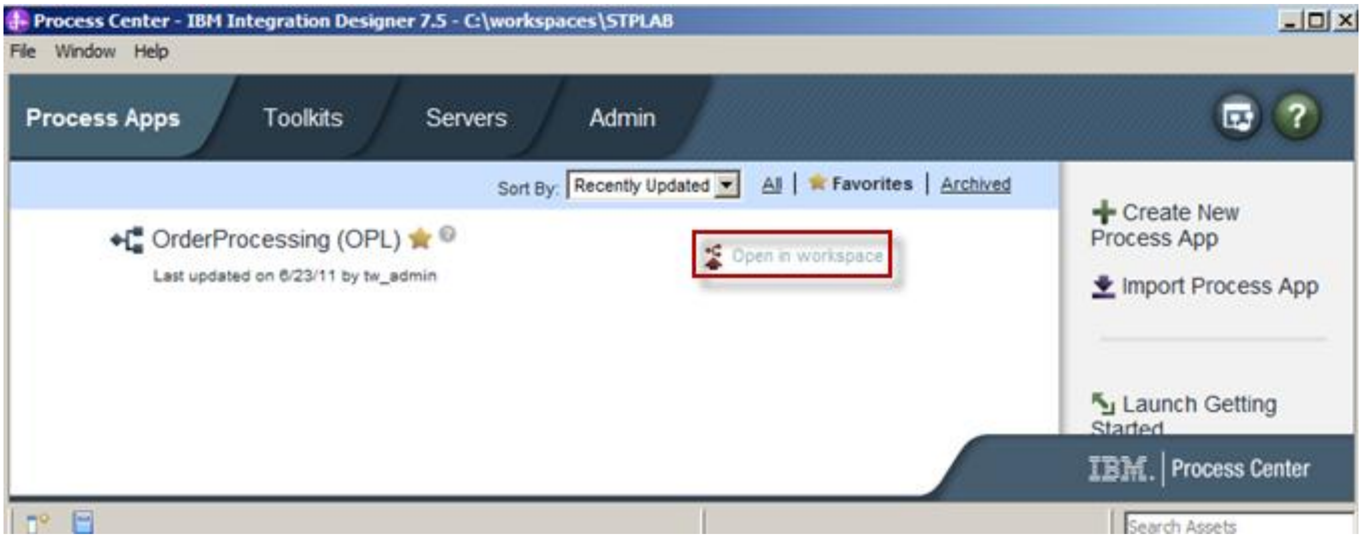
Process App Name: OrderProcessing
Acronym: OPL
Description: Order Processing Lab

__5. The process application's name can be anything you like.
  __a. Enter **OrderProcessing**.
  __b. The process acronym must be unique and is limited to seven characters. Enter **OPL**.
  __c. The Description is optional. Enter **Order Processing Lab**
  __d. Click the **Create** button

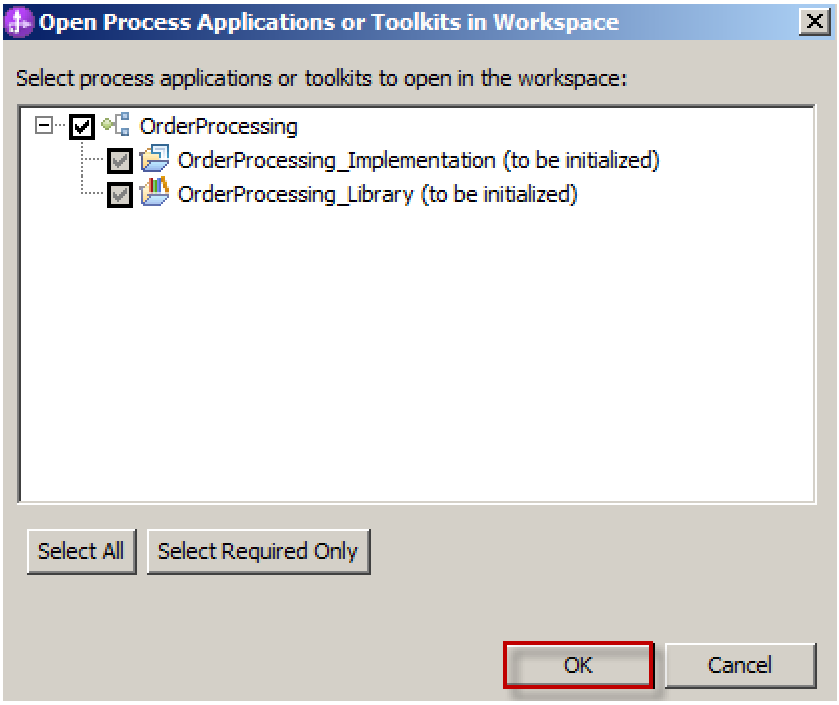
Process Acronyms

The acronym for a process application must be unique and is limited to seven characters (you are prevented from entering invalid characters). Advanced process authors can use the acronym as a unique identifier to distinguish reusable assets that share a common name

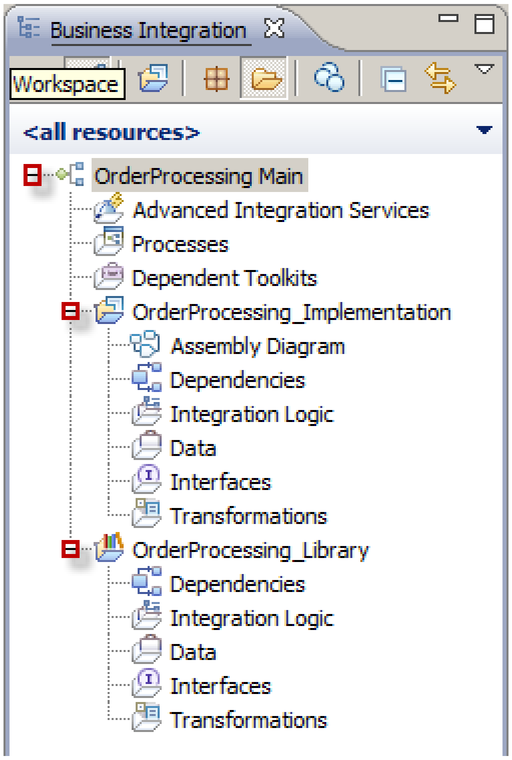
__6. By clicking the link **Open in workspace,** the new process application will displayed in IID



__7. You will be asked to select the applications and toolkits you want to open. Accept the defaults (the implementation and library should be pre-selected) and then click **OK.**


Open Process Applications or Toolkits in Workspace
Select process applications or toolkits to open in the workspace:
☑ OrderProcessing
☑ OrderProcessing_Implementation (to be initialized)
☑ OrderProcessing_Library (to be initialized)
Select All
Select Required Only
OK
Cancel

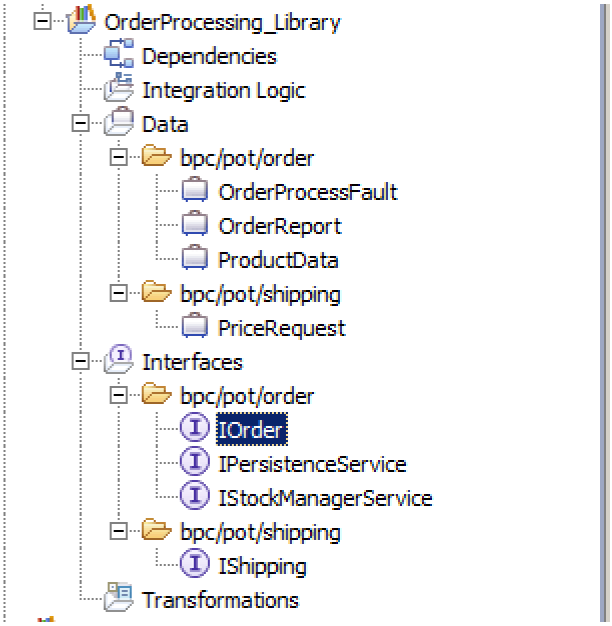
__8. The Business Integration perspective will be displayed and notice, it has created a library module and a business module (appended _Implementation) which are now part of the OrderProcessing Main project.



1.3 Populating the Library Module

Often, interfaces, business objects, business object maps, events, relationships, roles, and Web service ports need to be shared so that resources in several modules can use them. The *library* is a project that is used to store these resources; contents in a library project can be shared.

When you finish populating the library module you will have created four business objects and four interfaces that will be used by the Business Module. You will also have made your library module a dependant piece of your Business Module.



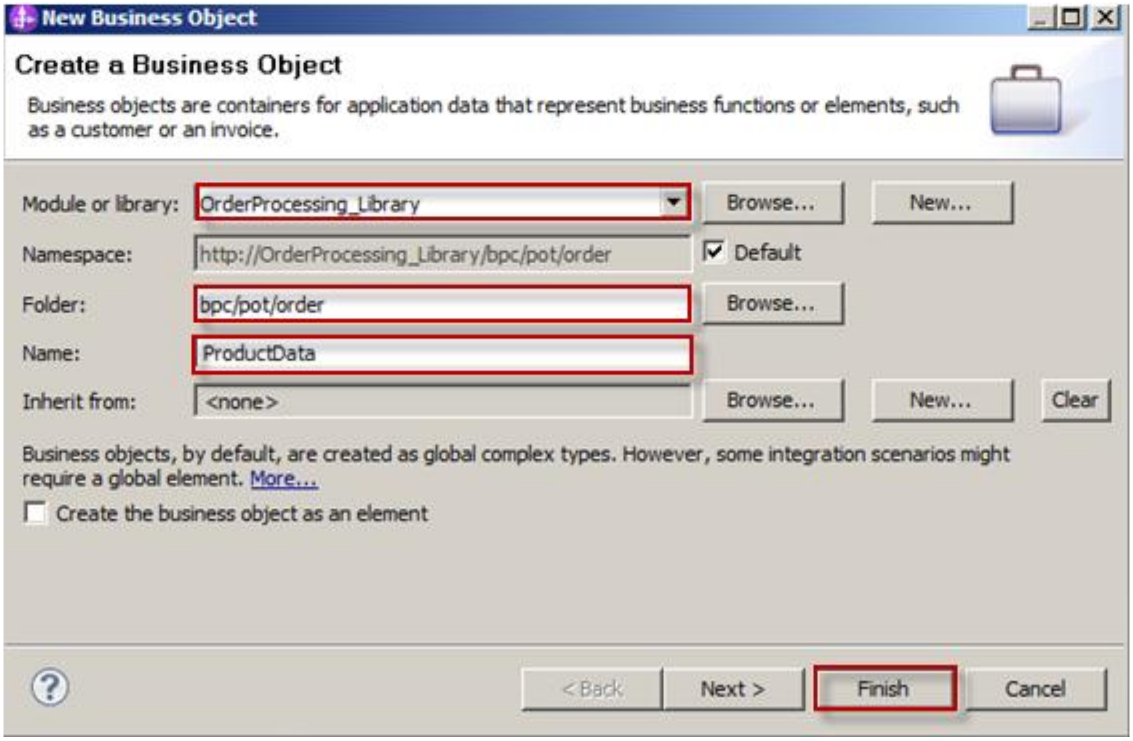
- OrderProcessing_Library
  - Dependencies
  - Integration Logic
  - Data
    - bpc/pot/order
      - OrderProcessFault
      - OrderReport
      - ProductData
    - bpc/pot/shipping
      - PriceRequest
  - Interfaces
    - bpc/pot/order
      - IOrder
      - IPersistenceService
      - IStockManagerService
    - bpc/pot/shipping
      - IShipping
  - Transformations

1.3.1 Create Business Objects

Business objects are one of the primary building blocks of any business integration solution. Essentially, business objects are containers for application data that represent business functions or elements, such as a customer or an invoice. A business object contains attributes, each of which has a name, a type (scalar type or another business object), a default value (for scalar types) and cardinality. Business objects can extend (define a superset of attributes) other business objects through parent/child relationships, however, a business object can only inherit from a single parent. These objects can also be used in conjunction with each other to perform a desired task.

__1. **Create Business Object - ProductData**

- __a. Expand the **OrderProcessing_Library** folder, highlight **Data**, right-mouse click, choose **New→Business Object**
- __b. In the folder field enter **bpc/pot/order**
- __c. In the name field enter **ProductData**
- __d. Click **Finish.**



Create a Business Object

Business objects are containers for application data that represent business functions or elements, such as a customer or an invoice.

Module or library: OrderProcessing_Library

Namespace: http://OrderProcessing_Library/bpc/pot/order ☑ Default

Folder: bpc/pot/order

Name: ProductData

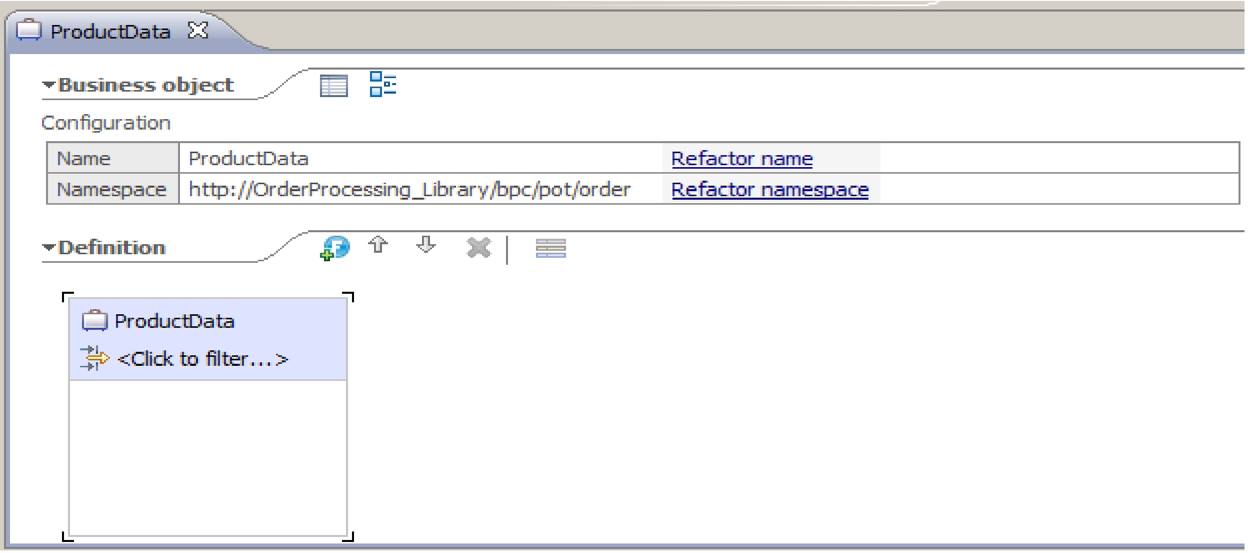
Inherit from: <none>

Business objects, by default, are created as global complex types. However, some integration scenarios might require a global element. More...

☐ Create the business object as an element

__2. The Business Object Editor is displayed.

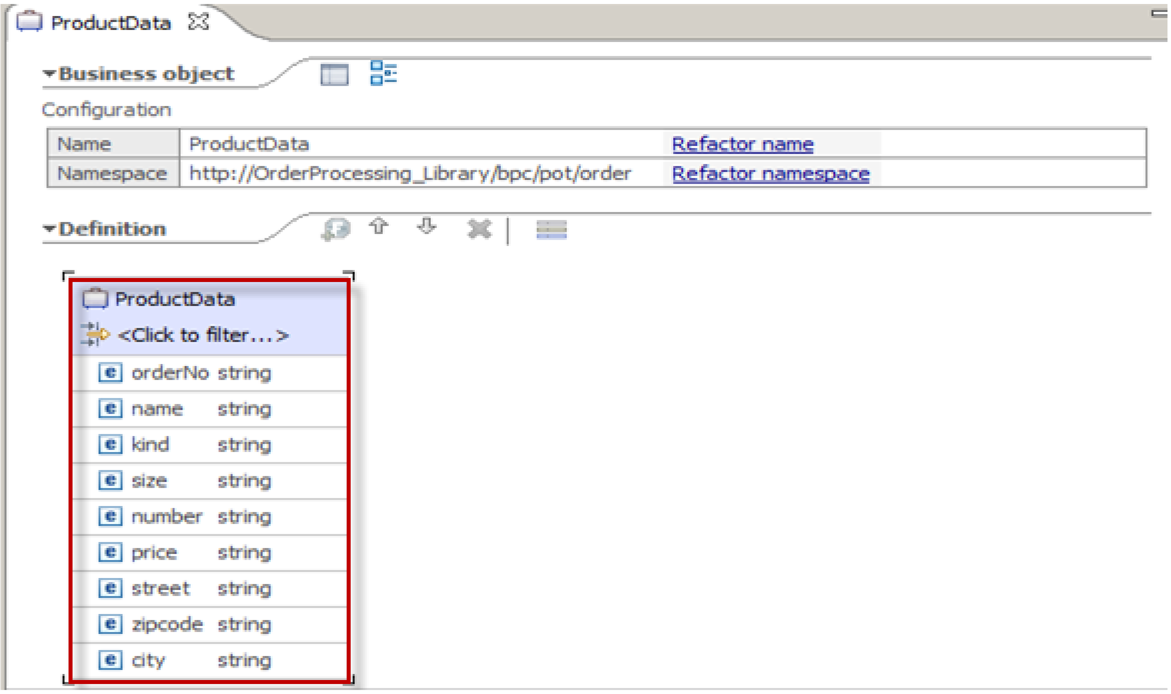
- __a. Add fields to the business object by clicking the **Add Field** button
- __b. Change the field name to **orderNo** leave the field type **string**



Configuration

Name	ProductData	Refactor name
Namespace	http://OrderProcessing_Library/bpc/pot/order	Refactor namespace

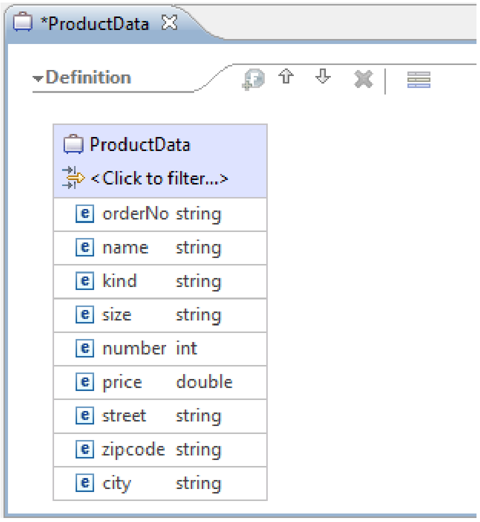
__c. Similarly add 7 more fields. Change the field names and types as shown.
- __i. Change the field name to **name** leave the type **string**
- __ii. Change the field name to **kind** leave the type **string**
- __iii. Change the field name to **size** leave the type **string**
- __iv. Change the field name to **number** change the type **int**
- __v. Change the field name to **price** leave the type to **string**
- __vi. Change the field name to **street** leave the type **string**
- __vii. Change the field name to **zipcode** leave the type **string**
- __viii. Change the field name to **city** leave the type **string**



Configuration

Name	ProductData	Refactor name
Namespace	http://OrderProcessing_Library/bpc/pot/order	Refactor namespace

ProductData	
orderNo	string
name	string
kind	string
size	string
number	string
price	string
street	string
zipcode	string
city	string



ProductData

<Click to filter...>

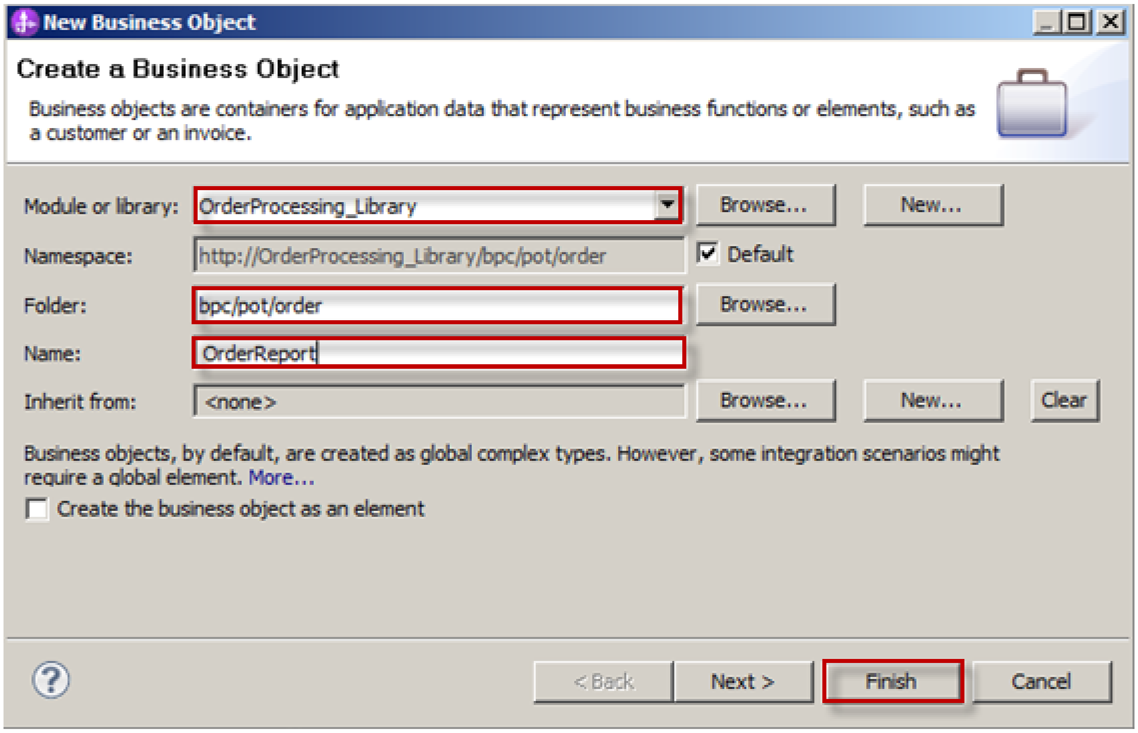
orderNo	string
name	string
kind	string
size	string
number	int
price	double
street	string
zipcode	string
city	string

__d. Click the **Save** button or **Ctrl+s** to save your work. You can close the business object.

__3. Create Business Object – OrderReport

You could import the remaining Business Objects (XSDs). Look for the *STPLab_XSDs.zip* in the lab files folder. In the navigation tree go to OrderProcessing_Library->Data->bpc/pot/order. Right-mouse click, choose Import. Select Business Integration->WSDL and XSD (Local), browse for the 3 XSD files and import them. **WARNING: you might have to refactor your project.**

__a. Or, similarly create another business object (**OrderProcessing_Library** > **Data > New→Business Object**) called OrderReport
__b. In the folder field enter **bpc/pot/order**
__c. In the name field enter **OrderReport**



Create a Business Object

Business objects are containers for application data that represent business functions or elements, such as a customer or an invoice.

Module or library: OrderProcessing_Library
Namespace: http://OrderProcessing_Library/bpc/pot/order ☑ Default
Folder: bpc/pot/order
Name: OrderReport
Inherit from: <none>

Business objects, by default, are created as global complex types. However, some integration scenarios might require a global element. More...

☐ Create the business object as an element __d. Add the following 7 fields to the business object by clicking the **Add Field** button

__e. Change the field name to **orderNo** leave the field type **string**

__f. Change the field name to **refNo** leave the type **string**

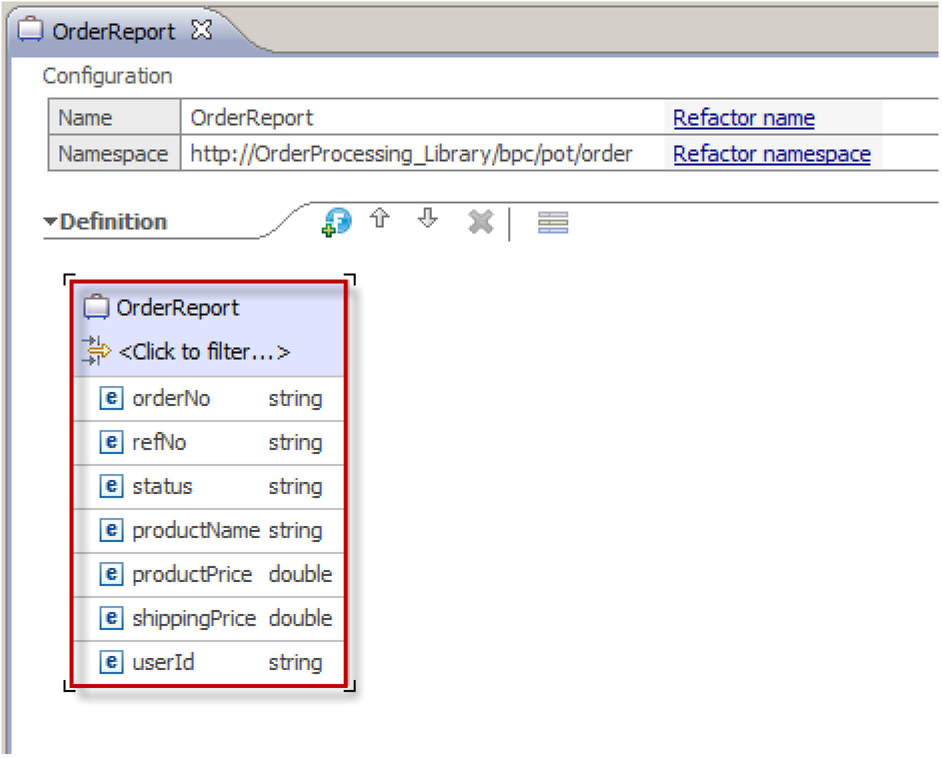
__g. Change the field name to **status** leave the type **string**

__h. Change the field name to **productName** leave the type **string**

__i. Change the field name to **productPrice** change the type to **double**

__j. Change the field name to **shippingPrice** change the type to **double**

__k. Change the field name to **userId** leave the type **string**



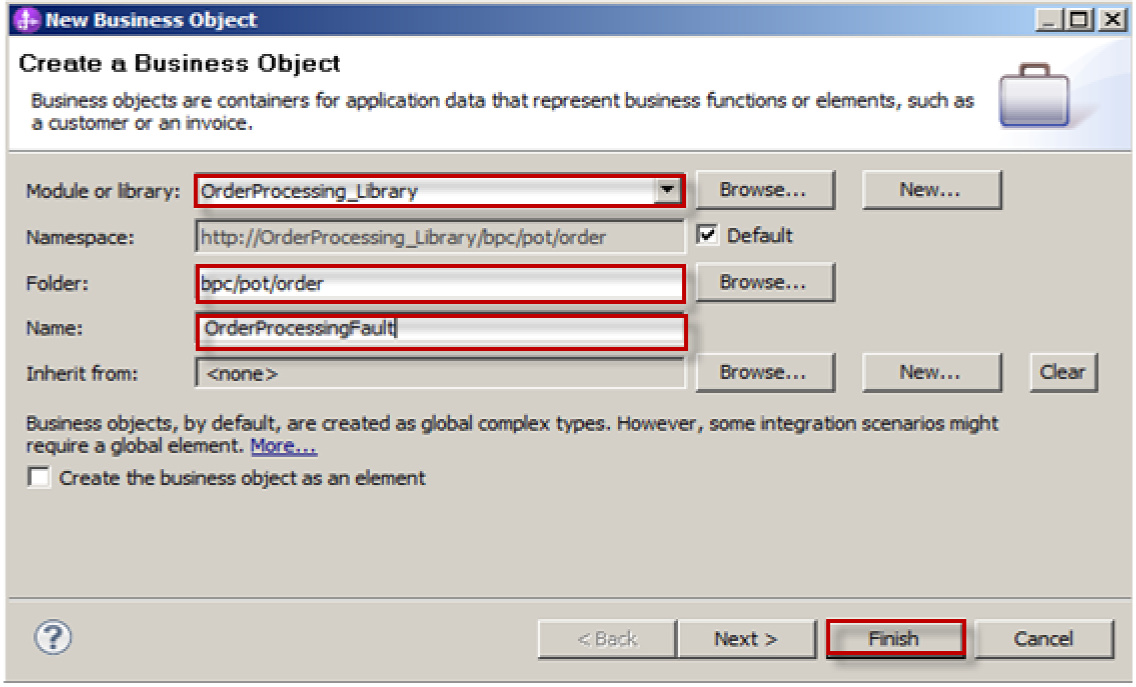
Name	OrderReport	Refactor name
Namespace	http://OrderProcessing_Library/bpc/pot/order	Refactor namespace

orderNo	string
refNo	string
status	string
productName	string
productPrice	double
shippingPrice	double
userId	string

__l. Click the **Save** button **or Ctrl+s** to save your work. You can close the business object.

__4. **Create Business Object – OrderProcessingFault**

__a. Similarly create another business object (**OrderProcessing_Library** > **Data > New→Business Object**) called OrderProcessingFault
__b. In the folder field enter **bpc/pot/order**
__c. In the name field enter **OrderProcessingFault**



New Business Object

Create a Business Object

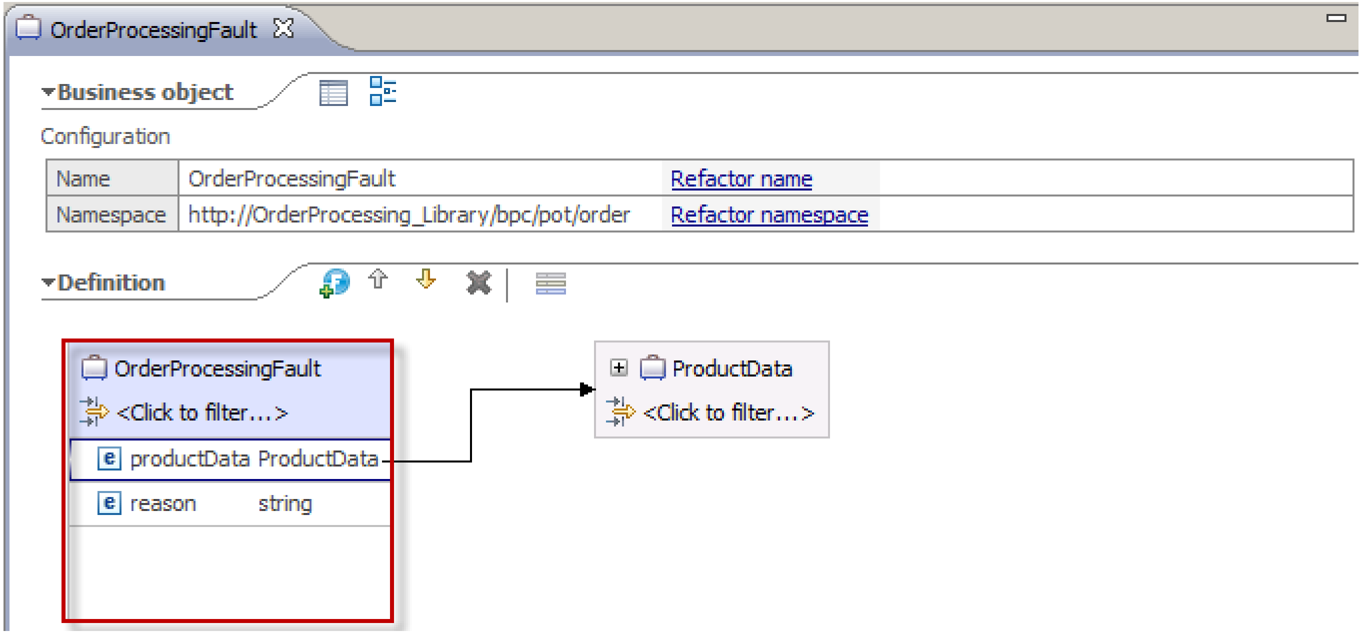
Business objects are containers for application data that represent business functions or elements, such as a customer or an invoice.

Module or library: OrderProcessing_Library
Namespace: http://OrderProcessing_Library/bpc/pot/order ☑ Default
Folder: bpc/pot/order
Name: OrderProcessingFault
Inherit from: <none>

Business objects, by default, are created as global complex types. However, some integration scenarios might require a global element. More...

☐ Create the business object as an element

__d. Add the following 2 fields to the business object by clicking the **Add Field** button. Notice one of the fields will use one of the new types – ProductData.
__e. Change the field name to **productData** change the field type **ProductData**
__f. Change the field name to **reason** leave the type **string**



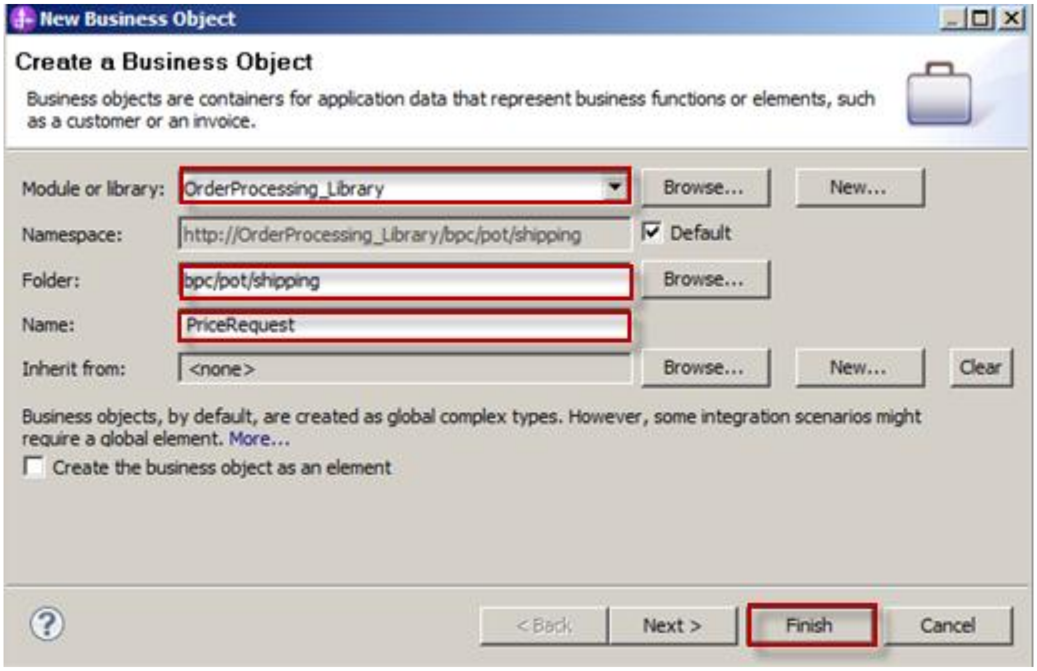
Configuration

Name	OrderProcessingFault	Refactor name
Namespace	http://OrderProcessing_Library/bpc/pot/order	Refactor namespace

__g. Click the **Save** button **or Ctrl+s** to save your work. You can close the business object.

__5. Create Business Object – PriceRequest

__a. Similarly create another business object (**OrderProcessing_Library** > **Data** > **New** > **BusinessObject**), called PriceRequest.
__b. In the folder field enter **bpc/pot/shipping**
__c. In the name field enter **PriceRequest**



Create a Business Object

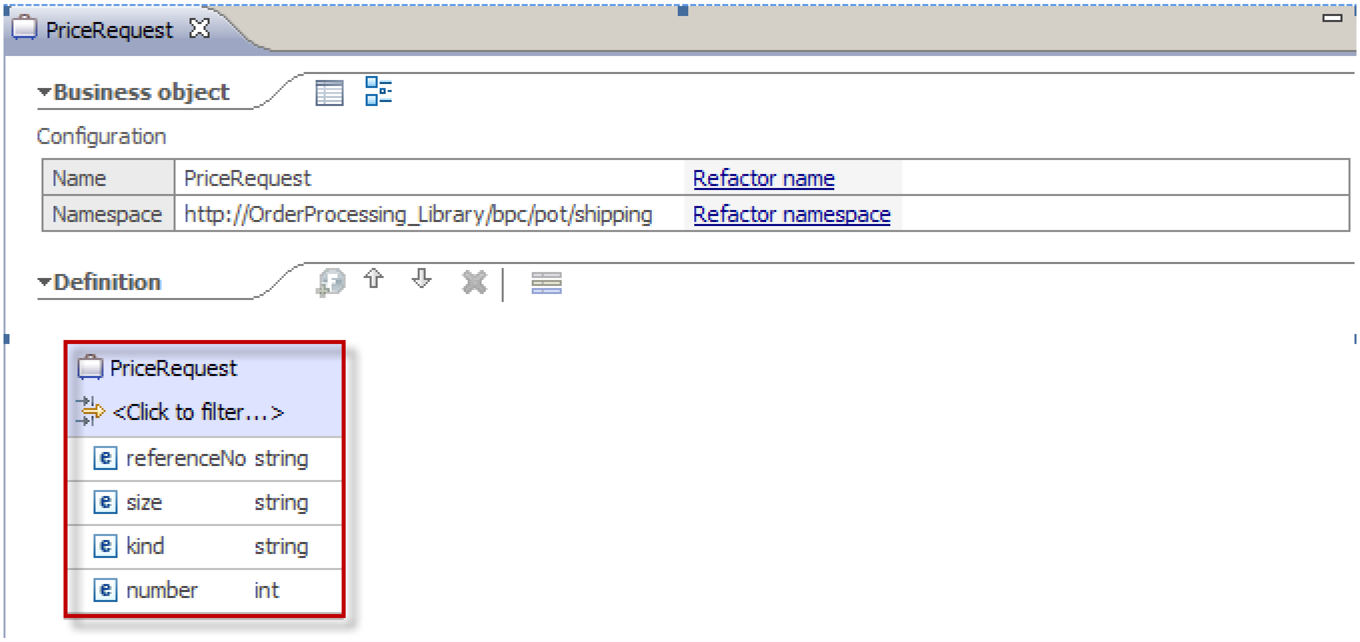
Business objects are containers for application data that represent business functions or elements, such as a customer or an invoice.

Module or library: OrderProcessing_Library
Namespace: http://OrderProcessing_Library/bpc/pot/shipping ☑ Default
Folder: bpc/pot/shipping
Name: PriceRequest
Inherit from: <none>

Business objects, by default, are created as global complex types. However, some integration scenarios might require a global element. More...

☐ Create the business object as an element

__d. Add the following 4 fields to the business object by clicking the **Add Field** button
__e. Change the field name to **referenceNo** leave the field type **string**
__f. Change the field name to **size** leave the type **string**
__g. Change the field name to **kind** leave the type **string**
__h. Change the field name to **number** change the type to **int**



Configuration

Name	PriceRequest	Refactor name
Namespace	http://OrderProcessing_Library/bpc/pot/shipping	Refactor namespace

PriceRequest

<Click to filter…>

referenceNo	string
size	string
kind	string
number	int

__i. Click the **Save** button **or Ctrl+s** to save your work. You can close the business object.

1.3.2 Create Interfaces

What is an interface? An interface provides the input and output details of a component. It is created independent of the internal implementation of the component. Every business process is also a Web service operation. A client of the process can initiate the process by invoking that operation. An interface in IID is an abstract Web service defined in a WSDL file (called portType). A business process must implement an operation of that abstract Web service or interface.

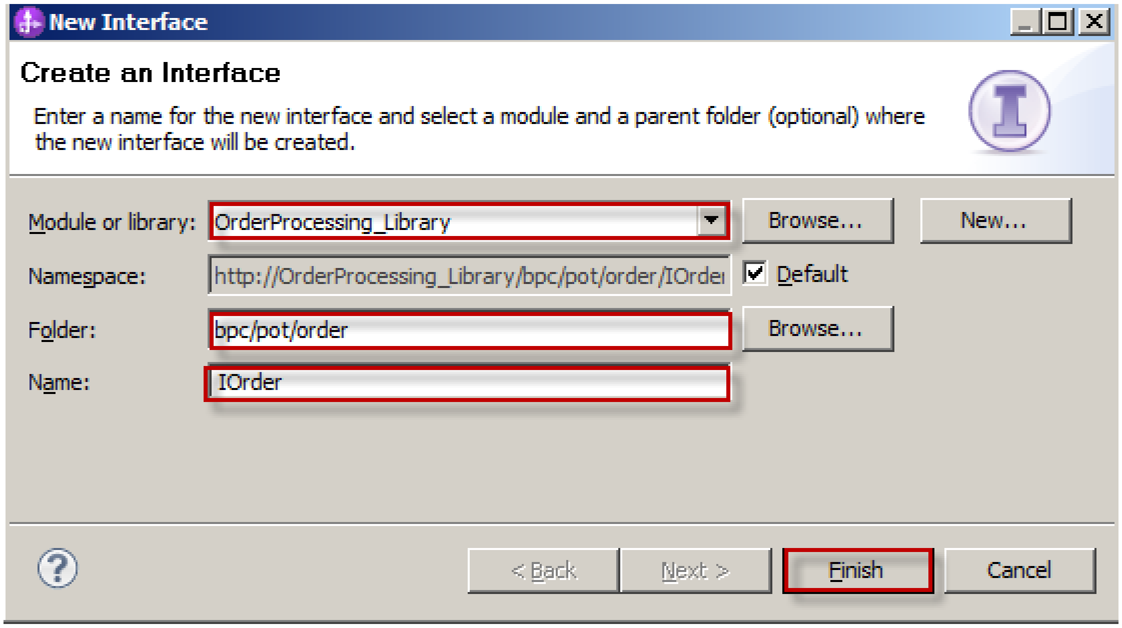
__1. Create Interface – IOrder

__a. Create a new interface by expanding the **OrderProcessing_Library** folder, highlight **Interfaces**, right-mouse click, choose **New→Interface**

__b. In the folder field Enter **bpc/pot/order**

__c. In the name field Enter **IOrder**

__d. Click **Finish**


New Interface

Create an Interface

Enter a name for the new interface and select a module and a parent folder (optional) where the new interface will be created.

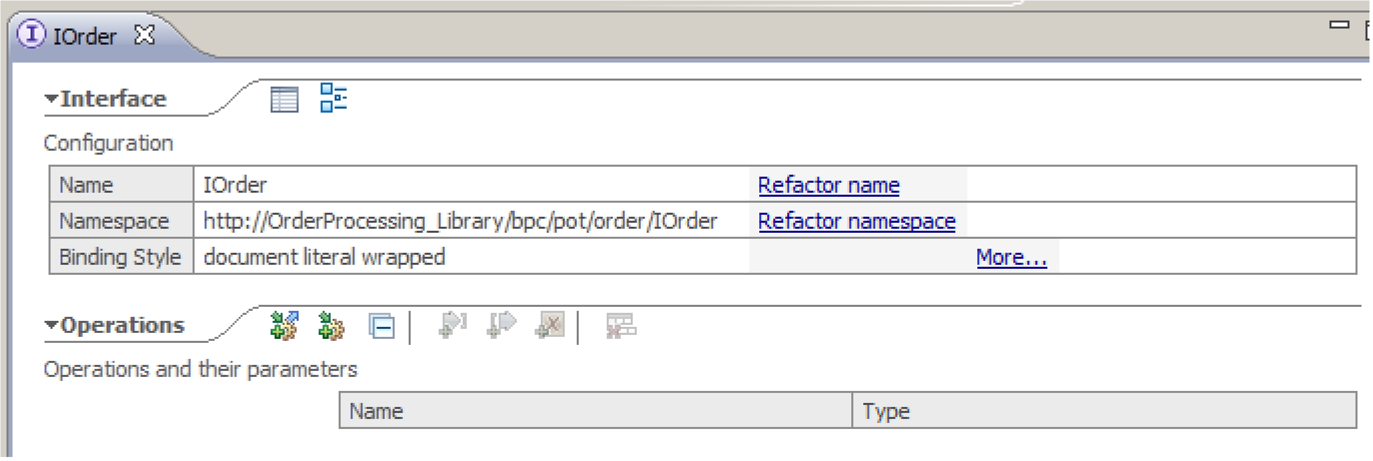
Module or library: OrderProcessing_Library

Namespace: http://OrderProcessing_Library/bpc/pot/order/IOrde ☑ Default

Folder: bpc/pot/order

Name: IOrder

__6. The Interface Editor is displayed. You can create a Request Response operation or a One Way operation.



Configuration

Name	IOrder	Refactor name
Namespace	http://OrderProcessing_Library/bpc/pot/order/IOrder	Refactor namespace
Binding Style	document literal wrapped	More…

Operations and their parameters

Name	Type

__e. Create a Request Response Operation by clicking the **Add Request Response Operation** icon

__f. Click on **operation1** and change its name from **operation1** to **processOrder**

__g. Under Inputs for the **processOrder** operation, click on **input1** and change the name to **productData.**

__h. Change its type from **string** to **ProductData** by clicking on the **string** link and selecting **ProductData** from the pop-up.

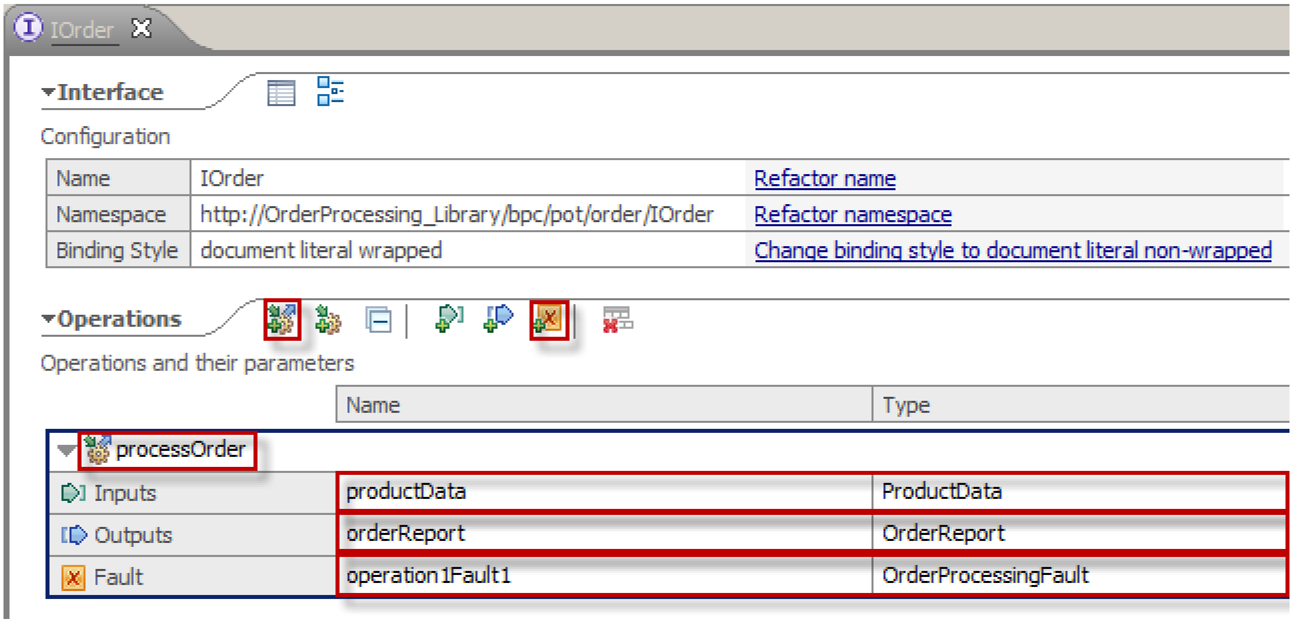
__i. Under Outputs for the **processOrder** operation, click on **output1** and change the name to **orderReport**.

__j. Change its type from **string** to **OrderReport** by clicking on the **string** link and selecting **OrderReport** from the pop-up.

__k. Create a fault by clicking the **Add Fault** icon

__l. Under fault for the **processOrder** operation, click on **operation1Fault** and change the name to **OrderProcessingFault**.

__m. Change its type from **string** to **OrderProcessingFault** by clicking on the **string** link and selecting **OrderProcessingFault** from the pop-up.

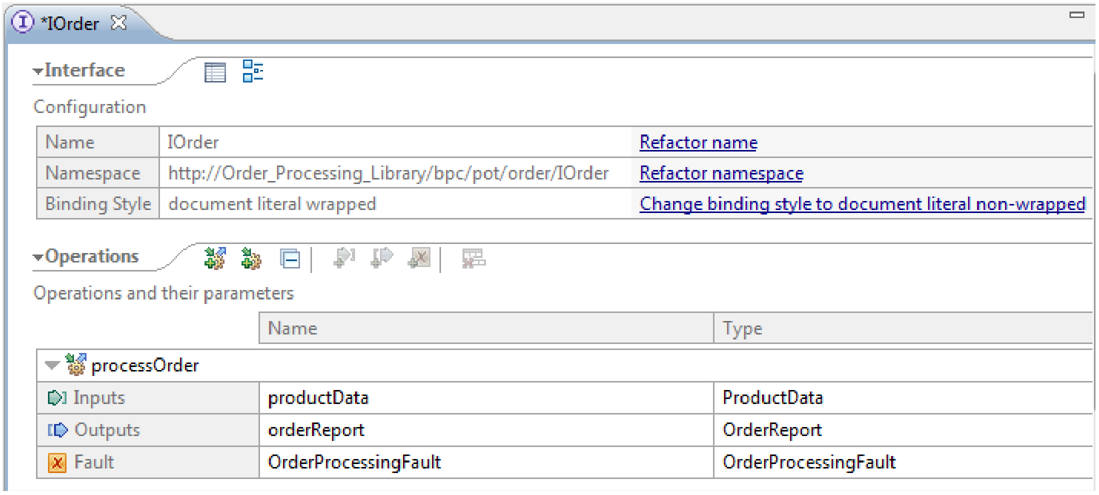


Configuration

Name	IOrder	Refactor name
Namespace	http://OrderProcessing_Library/bpc/pot/order/IOrder	Refactor namespace
Binding Style	document literal wrapped	Change binding style to document literal non-wrapped

Operations and their parameters

	Name	Type
processOrder		
Inputs	productData	ProductData
Outputs	orderReport	OrderReport
Fault	operation1Fault1	OrderProcessingFault



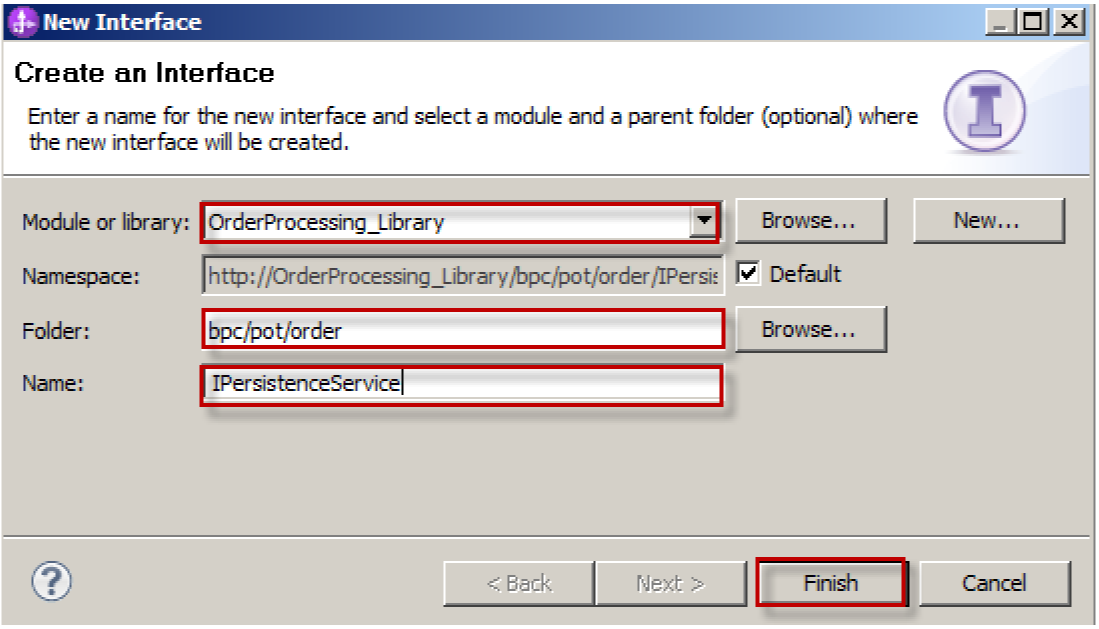
Name	IOrder	Refactor name
Namespace	http://Order_Processing_Library/bpc/pot/order/IOrder	Refactor namespace
Binding Style	document literal wrapped	Change binding style to document literal non-wrapped

	Name	Type
processOrder		
Inputs	productData	ProductData
Outputs	orderReport	OrderReport
Fault	OrderProcessingFault	OrderProcessingFault

__n. Click the **Save** button **or Ctrl+s** to save your work. You may close the interface editor.

__2. Create an Interface – IPersistenceService

__a. Similarly, create another interface (**OrderProcessing_Library** > **Interfaces** > **New** > **Interface)**
__b. In the folder field Enter **bpc/pot/order**
__c. In the name field Enter **IPersistenceService**



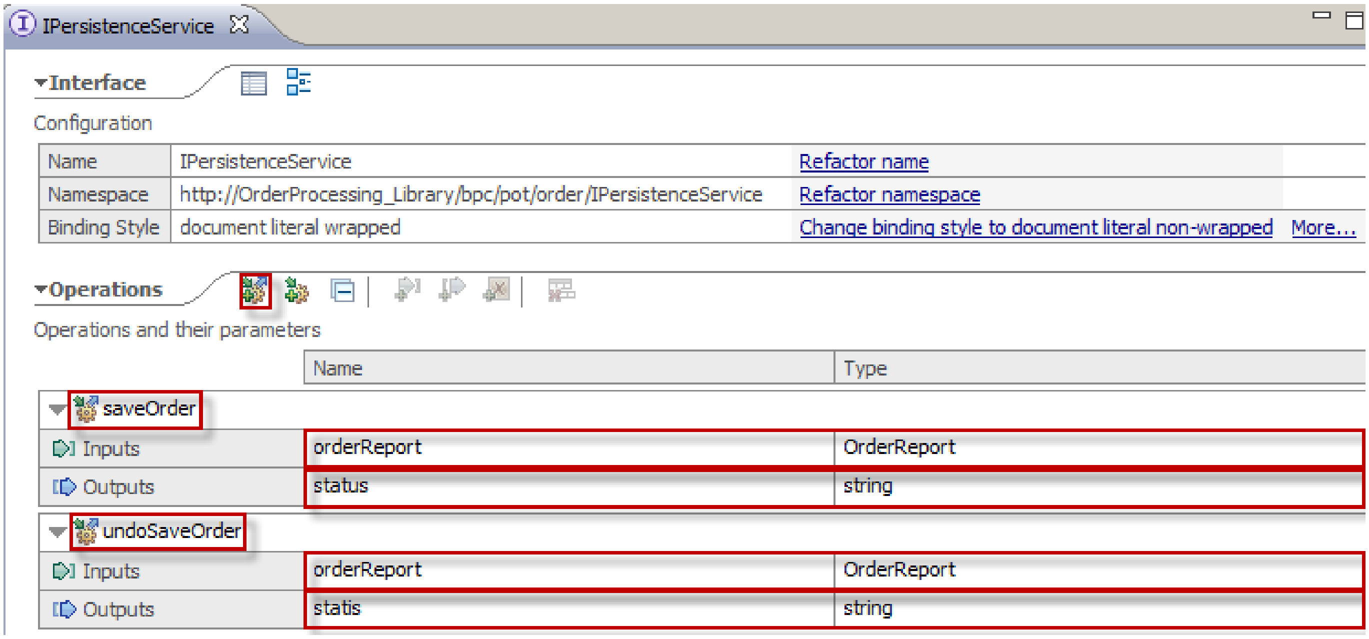
Create an Interface

Enter a name for the new interface and select a module and a parent folder (optional) where the new interface will be created.

Module or library: OrderProcessing_Library
Namespace: http://OrderProcessing_Library/bpc/pot/order/IPersis ☑ Default
Folder: bpc/pot/order
Name: IPersistenceService

__d. This interface will have 2 operations:
  __i. Create a Request Response Operation operation by clicking the **Add Request Response Operation** icon
  __**ii.** Under Inputs, change the name from **operation1** to **saveOrder**
  __iii. Change the name of **input1** to **orderReport.**
  __iv. Change its type from **string** to **OrderReport**.
  __v. Under Outputs, change the name from **output1** to **status**. Leave the type as **string**.

  __vi. Create another Request Response Operation by clicking the **Add Request Response Operation** icon
  __**vii.** Change the name from **operation2** to **undoSaveOrder**
  __viii. Under Inputs change the name from **input1** to **orderReport.**
  __ix. Change its type from **string** to **OrderReport**.
  __x. Under outputs, change the name from **output1** to **status**. Leave the type as **string**.



Configuration

Name	IPersistenceService	Refactor name	
Namespace	http://OrderProcessing_Library/bpc/pot/order/IPersistenceService	Refactor namespace	
Binding Style	document literal wrapped	Change binding style to document literal non-wrapped	More…

Operations and their parameters

	Name	Type
saveOrder		
Inputs	orderReport	OrderReport
Outputs	status	string
undoSaveOrder		
Inputs	orderReport	OrderReport
Outputs	statis	string

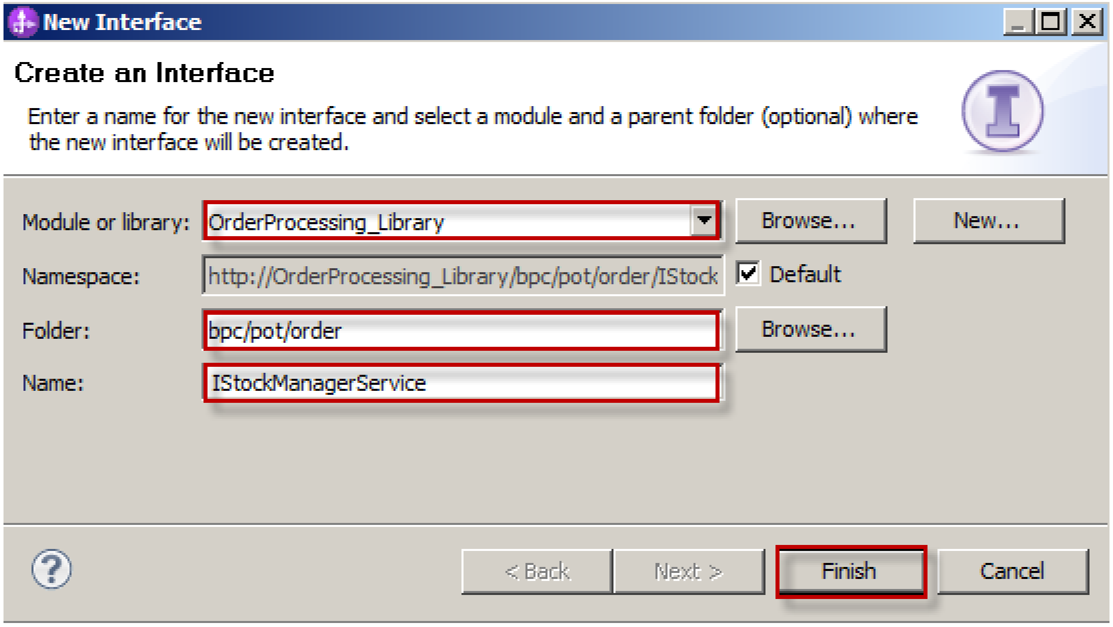
__e. Click the **Save** button **or Ctrl+s** to save your work. You can close the interface.

__3. Create Interface – IStockManagerService

__a. Similarly, create another interface (**OrderProcessingLibrary** > **Interfaces** > **New** > **Interface**), named IStockManagerService

__b. In the folder field Enter **bpc/pot/order**

__c. In the name field Enter **IStockManagerService**



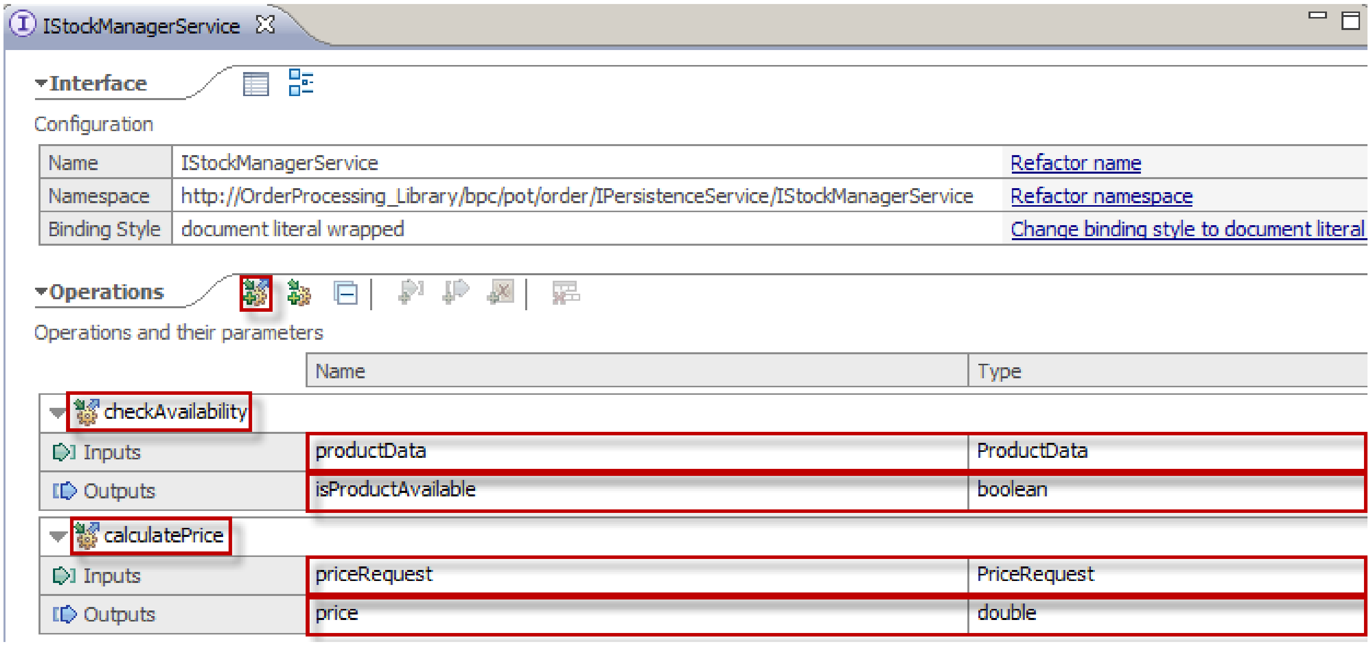
Enter a name for the new interface and select a module and a parent folder (optional) where the new interface will be created.

Module or library: OrderProcessing_Library
Namespace: http://OrderProcessing_Library/bpc/pot/order/IStock ☑ Default
Folder: bpc/pot/order
Name: IStockManagerService

__d. This interface will also have 2 operations:

- __i. Create a **Request Response Operation** by clicking the **Add Request Response Operation** icon
- __**ii.** Change the name of **operation1** to **checkAvailability**
- __iii. Change the name of **input1** to **productData.** Change its type from **string** to **ProductData**.
- __iv. Change the name of **output1** to **isProductAvailable.** Change its type from **string** to **boolean**.

- __v. Create a second **Request Response Operation** by clicking the **Add Request Response Operation** icon
- __**vi.** Change the name of **operation2** to **calculatePrice**
- __vii. Change the name of input1 to **priceRequest.** Change its type from **string** to **PriceRequest**.
- __viii. Change the name of **output1** to **price.** Change its type from **string** to **double**.



Configuration

Name	IStockManagerService	Refactor name
Namespace	http://OrderProcessing_Library/bpc/pot/order/IPersistenceService/IStockManagerService	Refactor namespace
Binding Style	document literal wrapped	Change binding style to document literal

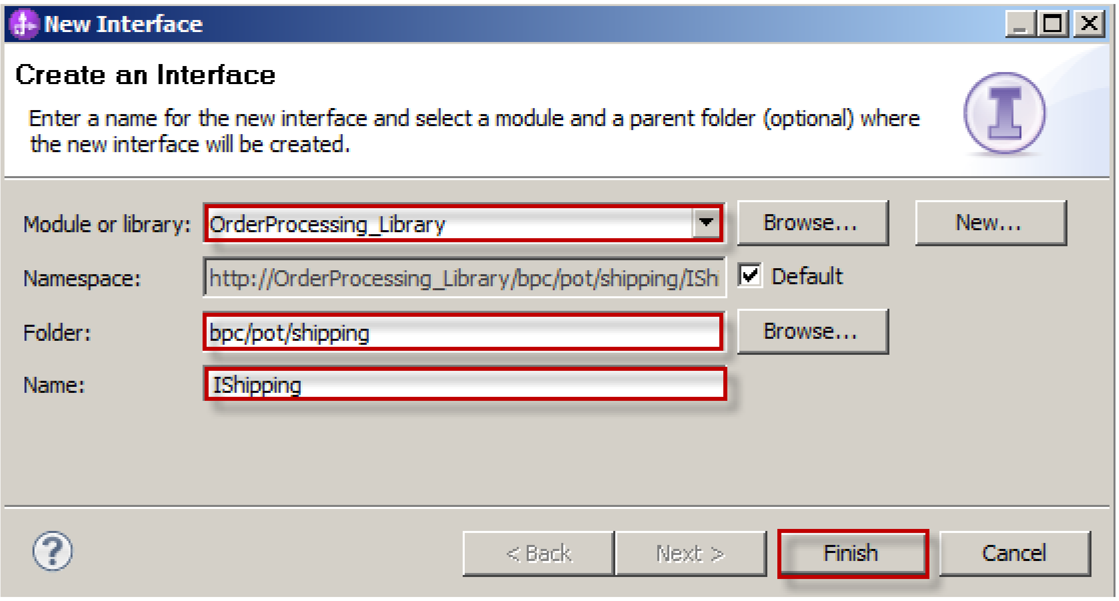
Operations and their parameters

	Name	Type
checkAvailability		
Inputs	productData	ProductData
Outputs	isProductAvailable	boolean
calculatePrice		
Inputs	priceRequest	PriceRequest
Outputs	price	double

__e. Click the **Save** button **or Ctrl+s** to save your work. You can close the interface.

__4. Create an Interface – IShipping

__a. Similarly create another interface (**OrderProcessing_Library** > **Interfaces** > **New** > **Interface)**
__b. In the folder field enter **bpc/pot/shipping**
__c. In the name field enter **IShipping**

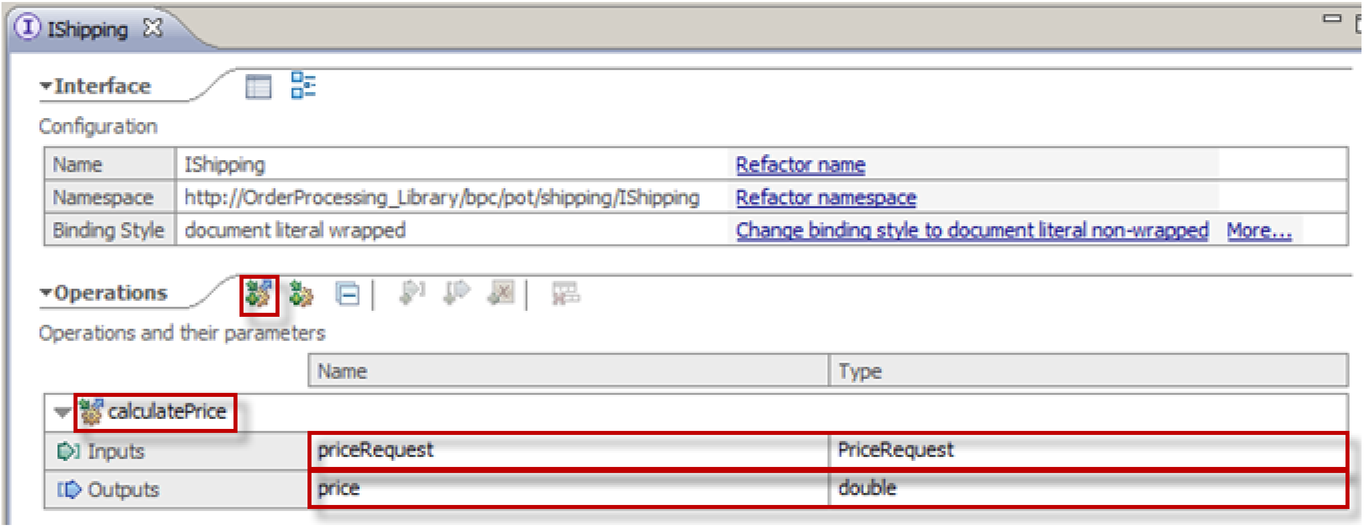


Create an Interface

Enter a name for the new interface and select a module and a parent folder (optional) where the new interface will be created.

Module or library: OrderProcessing_Library
Namespace: http://OrderProcessing_Library/bpc/pot/shipping/ISh ☑ Default
Folder: bpc/pot/shipping
Name: IShipping

__d. Create a **Request Response Operation** by clicking the **Add Request Response Operation** icon
__e. Change the name of **operation1** to **calculatePrice**
__f. Change the name of **input1** to **priceRequest.** Change its type from **string** to **PriceRequest**.
__g. Change the name of **output1** to **price**. Change its type from **string** to **double**.



Configuration

Name	IShipping	Refactor name
Namespace	http://OrderProcessing_Library/bpc/pot/shipping/IShipping	Refactor namespace
Binding Style	document literal wrapped	Change binding style to document literal non-wrapped More...

Operations and their parameters

	Name	Type
calculatePrice		
Inputs	priceRequest	PriceRequest
Outputs	price	double

__h. Click the **Save** button **or Ctrl+s** to save your work. You can close the interface.

1.4 Creating the Business Process

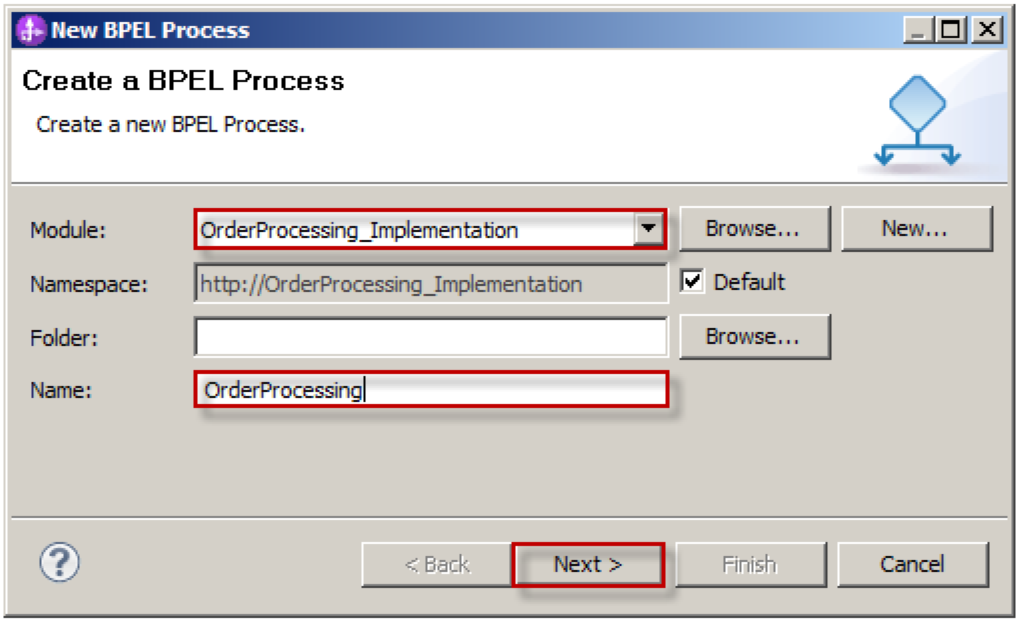
A *BPEL process* component implements a business process. Its implementation language is the industry standard Business Process Execution Language for Web Services (BPEL4WS) and its IBM extensions. A BPEL process implements a potentially long-running service through the use of more elementary services. A BPEL process created in the process editor can do the following things:

- Describe the orchestration of other services using control flow graphs
- Use variables to keep the process state
- Use sophisticated error handling through fault handling
- Support asynchronous events
- Correlate inbound requests with the right instance of a particular process by using correlation sets to mark that business data within the request that identifies the instance (for example, a customer ID)
- Provide extended transactions through sophisticated compensation support

In addition to these standard BPEL items, IBM Integration Developer also extends BPEL to include people into a process with *human task* support.

1.4.1 Create a BPEL Process

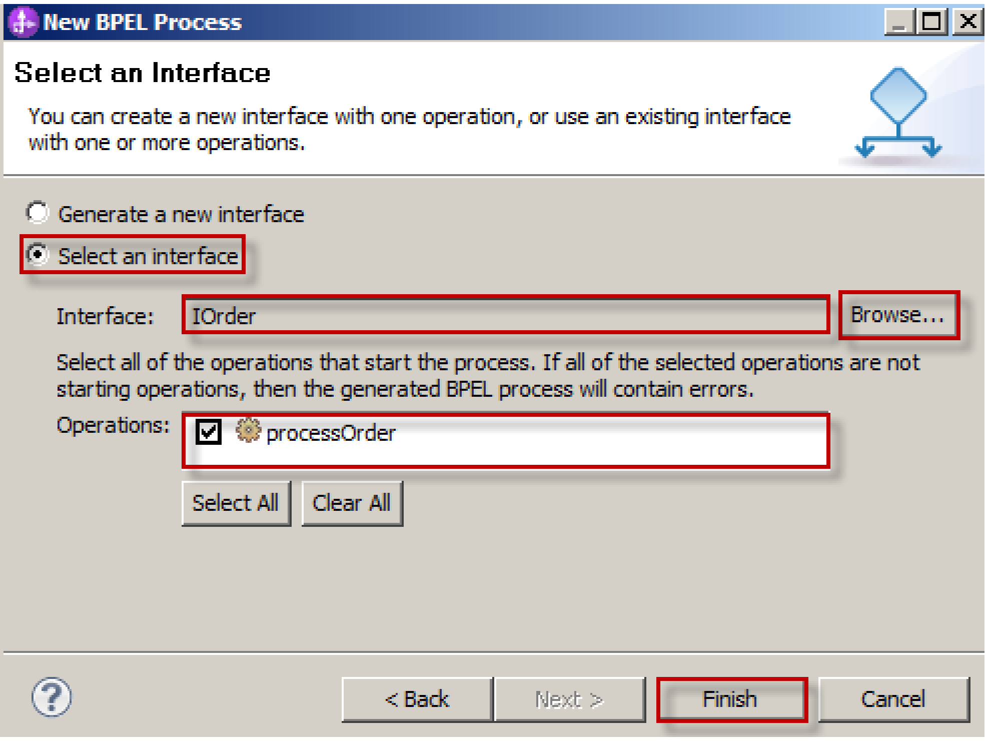
__1. Expand the **OrderProcessing_Implementation** and highlight the **Integration Logic** folder.
  __a. Right-click and select New→BPEL Process
  __b. In the Name field, enter **OrderProcessing**
  __c. Click **Next**



Module: OrderProcessing_Implementation
Namespace: http://OrderProcessing_Implementation ☑ Default
Folder:
Name: OrderProcessing

  __d. Select the **Microflow** radio button and click **Next**

__e. Click the radio button to **Select an interface**
__f. Click the **Browse...** button and choose the IOrder interface
__g. Click **OK**



New BPEL Process

Select an Interface

You can create a new interface with one operation, or use an existing interface with one or more operations.

○ Generate a new interface
● Select an interface

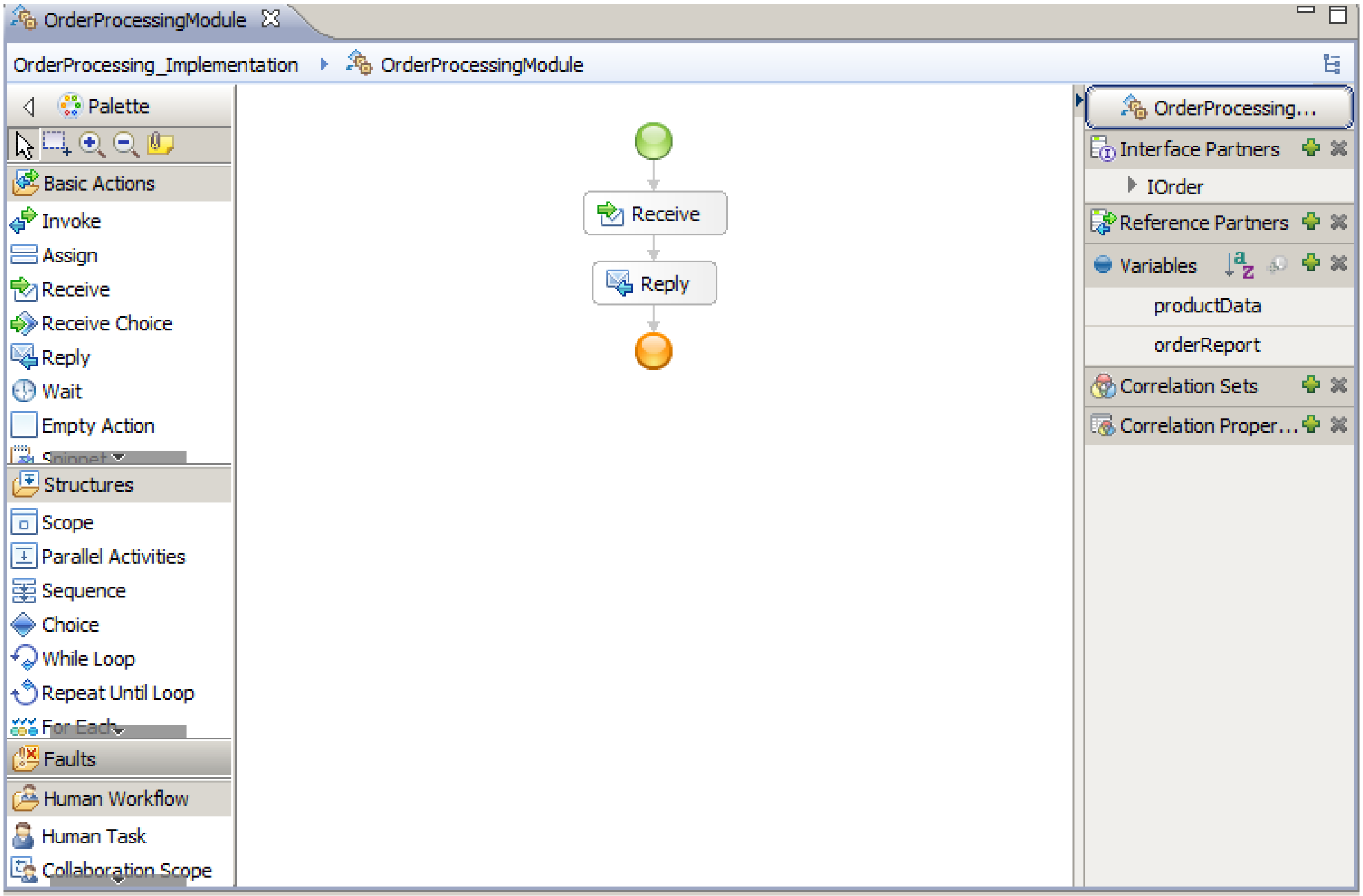
Interface: IOrder

Select all of the operations that start the process. If all of the selected operations are not starting operations, then the generated BPEL process will contain errors.

Operations: ☑ processOrder

__h. Make sure the checkbox next to the processOrder operation is checked
__i. Click **Finish**

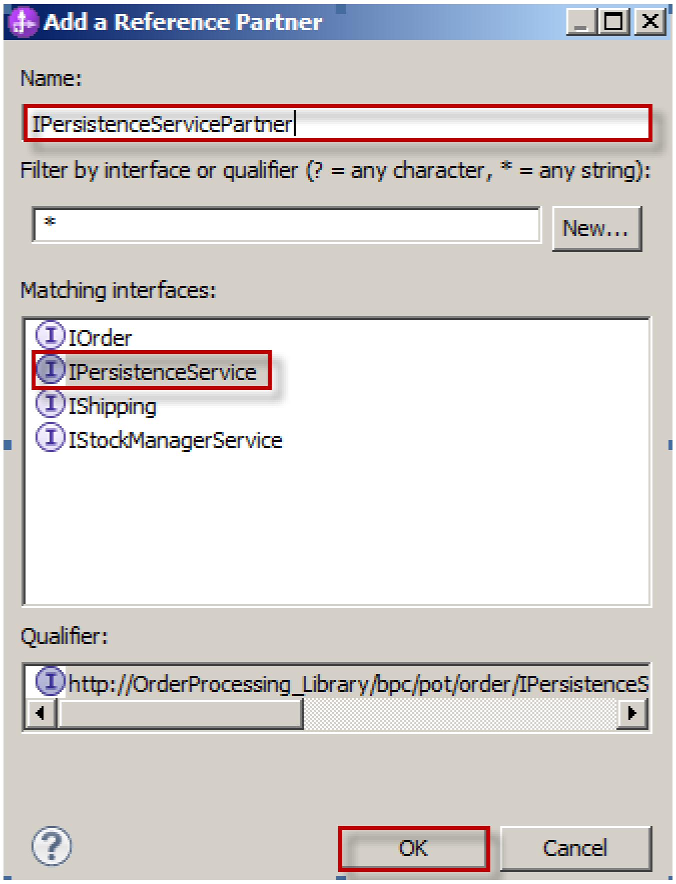
__2. The BPEL editor comes up displaying a skeleton process with a Receive and Reply node.



1.4.2 Add Reference Partners

__1. Add **IPersistenceServicePartner** reference partner

- __a. Click the ✚ icon on the Reference Partners ✚ ✖ to add a **Reference Partner**
- __b. Change the **Name** from partner to **IPersistenceServicePartner**.
- __c. Select the **IPersistenceService** interface from the list of **Matching Interfaces**
- __d. Click **OK**



Add a Reference Partner

Name: IPersistenceServicePartner

Filter by interface or qualifier (? = any character, * = any string): *

Matching interfaces:

- IOrder
- IPersistenceService
- IShipping
- IStockManagerService

Qualifier: http://OrderProcessing_Library/bpc/pot/order/IPersistenceS

__2. Add **IShippingPartner** reference partner

- __a. Click the ✚ icon on the Reference Partners ✚ ✖ to add a **Reference Partner**
- __b. Change the **Name** from partner to **IShippingPartner**
- __c. Select the **IShipping** interface from the list of **Matching Interfaces**
- __d. Click **OK**

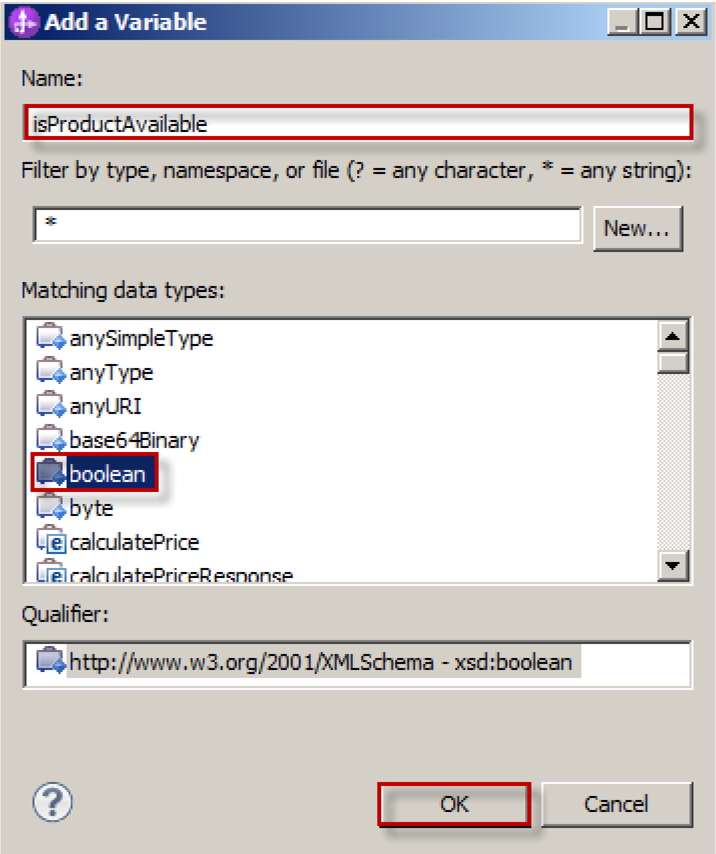
__3. Add **IStockManagerServicePartner** reference partner

- __a. Click the ✚ icon on the Reference Partners ✚ ✖ to add a **Reference Partner**
- __b. Change the Name from partner to **IStockManagerServicePartner**
- __c. Select the **IStockManagerService** interface from the list of **Matching Interfaces**
- __d. Click **OK**

1.4.3 Add Variables

__1. Add **isProductAvailable** variable

- __a. Click the ✚ icon on the ● Variables ✚ ✖



Name: isProductAvailable

Filter by type, namespace, or file (? = any character, * = any string): *

anySimpleType
anyType
anyURI
base64Binary
boolean
byte
calculatePrice
calculatePriceResponse

Qualifier: http://www.w3.org/2001/XMLSchema - xsd:boolean

__b. Change the **Name** from Variable to **isProductAvailable**.
__**c.** Select the **boolean** type from the list of **Matching data types**
__d. Click **OK**

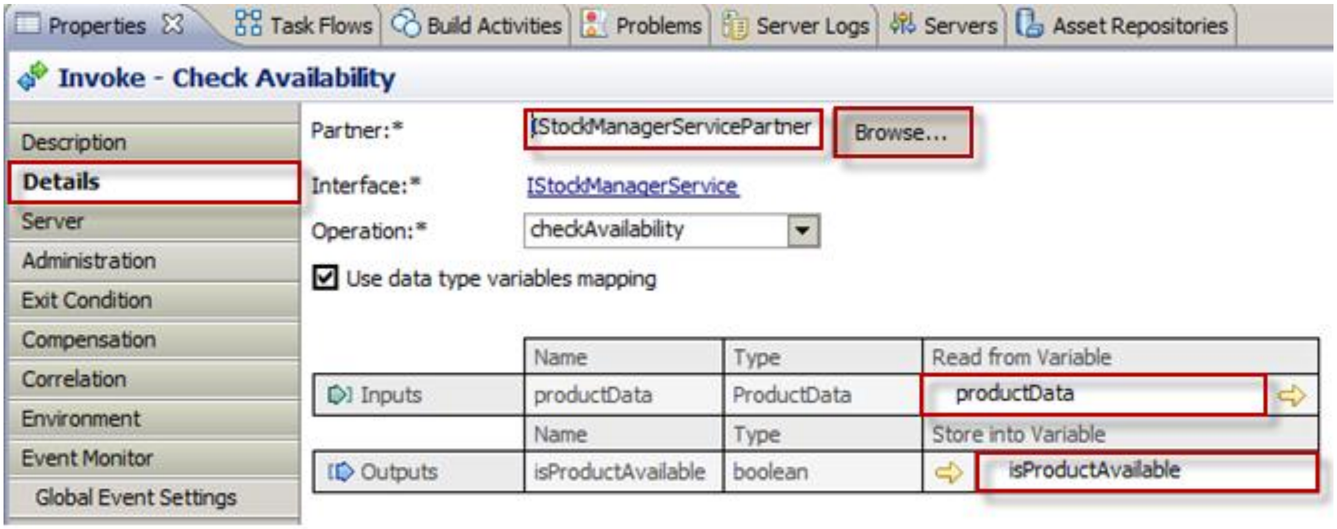
__2. Similarly add the following 5 variables:
  __**a.** **priceRequest** variable of type **PriceRequest**
  __**b.** **price**.variable of type **double**
  __**c.** **shippingPrice** variable of type **double**
  __d. **status** variable of type **string**
  __e. **orderProcessingFault** variable of type **OrderProcessingFault**

Click the **Save** button **or Ctrl+s** to save your work.

Constructing the BPEL Process

__1. Add an **Invoke** activity called **CheckAvailabiility**
  __a. From the Basic Actions palette on the left, drag an Invoke activity Invoke between the **Receive** and **Reply** nodes.
  __b. In the properties section change the **Name** from Invoke to **CheckAvailability** and change the **Display name** from Invoke to **Check Availability**.
  __c. In the **Details** tab add the partner by clicking the **Browse...** button and selecting **IStockManagerServicePartner** from the pop-up and click the **OK** button.
  __d. Map **productData** to variable **productData** by clicking the **(none)** link and selecting **productData** from the pop-up.

__e. Map **isProductAvailable** to variable **isProductAvailable** by clicking the **(none)** link and selecting **isProductAvailable** from the pop-up.



Invoke - Check Availability

Partner:* IStockManagerServicePartner

Interface:* IStockManagerService

Operation:* checkAvailability

☑ Use data type variables mapping

	Name	Type	Read from Variable
Inputs	productData	ProductData	productData
	Name	**Type**	**Store into Variable**
Outputs	isProductAvailable	boolean	isProductAvailable

__2. Add a **Choice** activity called **isProductAvailable**

__a. From the Structures palette, drag a Choice activity ◆ Choice between **CheckAvailability** and **Reply** node.

__b. In the properties section change the **Name:*** from Choice to **isProductAvailable** and change the **Display name:** to **Is Product Available?**

__3. Modify the Case node

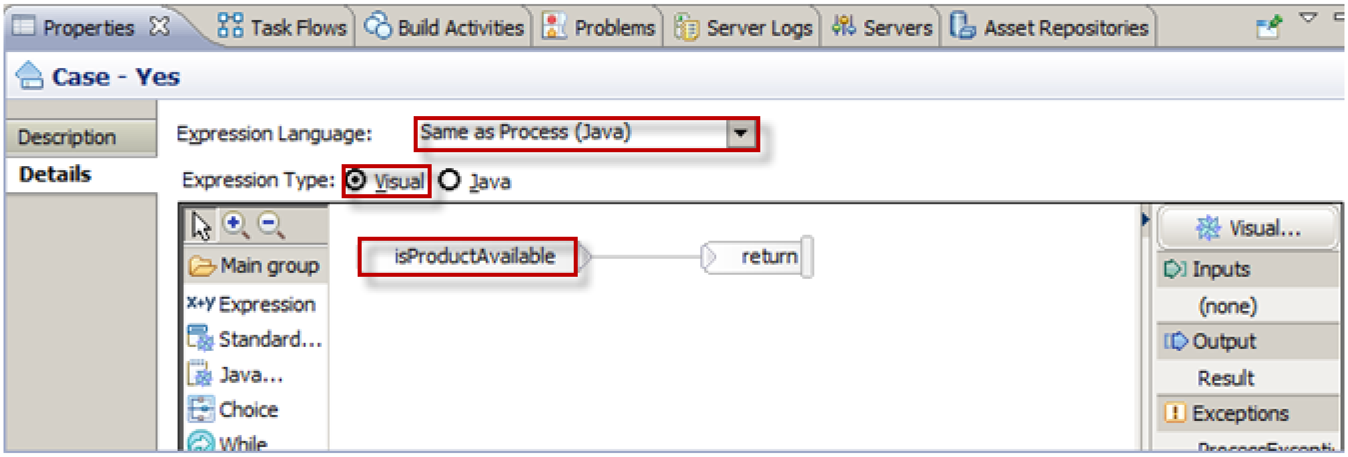
__a. The previous step automatically placed a **Case** activity within the Choice block. In the properties change the **Display name** from **Case** to **Yes.**

__b. Click the **Details** tab to create an expression

__c. Set the **Expression Language** to **Same as Process (Java)**.

__d. Choose the **Expression Type** to **Visual**

__e. Click the **true** link and select **isProductAvailable** from the pop-up menu.



Case - Yes

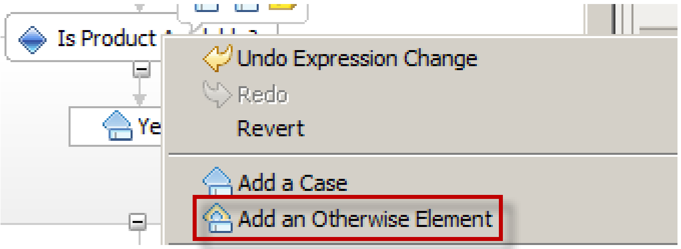
Expression Language: Same as Process (Java)

Expression Type: ● Visual ○ Java

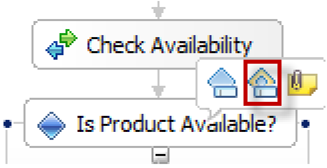
isProductAvailable → return

__4. Add an **Otherwise Element**

__a. In the BPEL diagram highlight the Choice activity named Is Product Available right-click and select **Add an Otherwise Element** from the pop-up; or you can click the icon to add the Otherwise element.

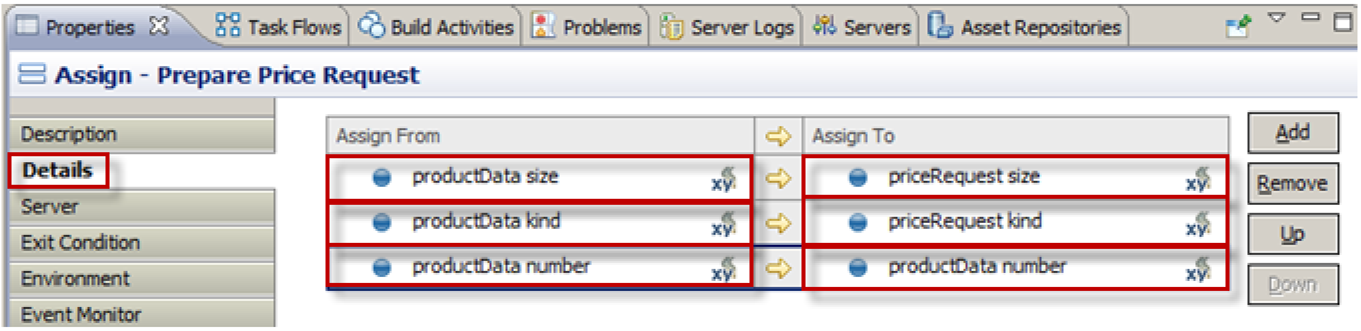


or



__5. Add an **Assign** activity

- __a. From the Basic Actions palette, drag an Assign Activity Assign under the Case action **Yes**
- __b. In the Properties, change the **Name** from **Assign** to **preparePriceRequest**
- __c. Change the **Display name** to **Prepare Price Request**
- __d. Click the **Details** tab and map the follow variables:
  - __i. Map productData.size to priceRequest.size
  - __ii. Map productData.kind to priceRequest.kind
  - __iii. Map productData.number to priceRequest.number

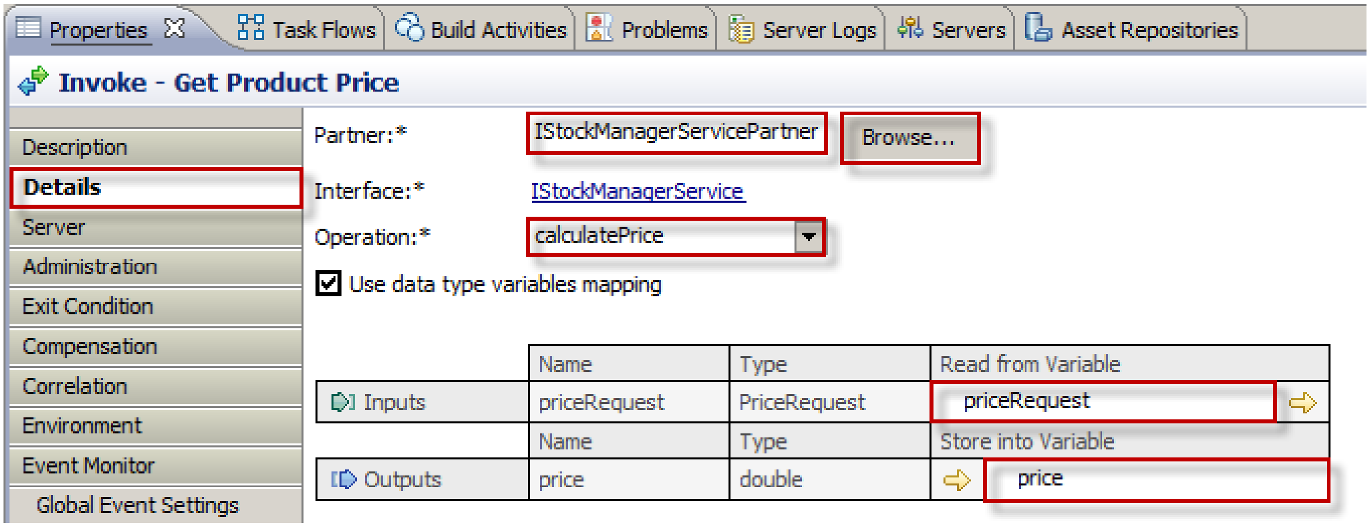


Assign - Prepare Price Request

Assign From		Assign To
productData size	⇨	priceRequest size
productData kind	⇨	priceRequest kind
productData number	⇨	productData number

__6. Add an **Invoke** activity

- __a. From the Basic Actions palette, drag an **Invoke Activity** under the **preparePriceRequest**.
- __b. In the properties change the **Name** from Invoke to **GetProductPrice**
- __c. Change the **Display name** to **Get Product Price**
- __d. Click the **Details** tab
- __e. To assign a Partner, click the **Browse...** button
- __f. Select the **IStockManagerServicePartner** from the pop-up
- __g. Click **OK**
- __h. Choose **calculatePrice** for the **Operation**



Invoke - Get Product Price

Partner:* IStockManagerServicePartner

Interface:* IStockManagerService

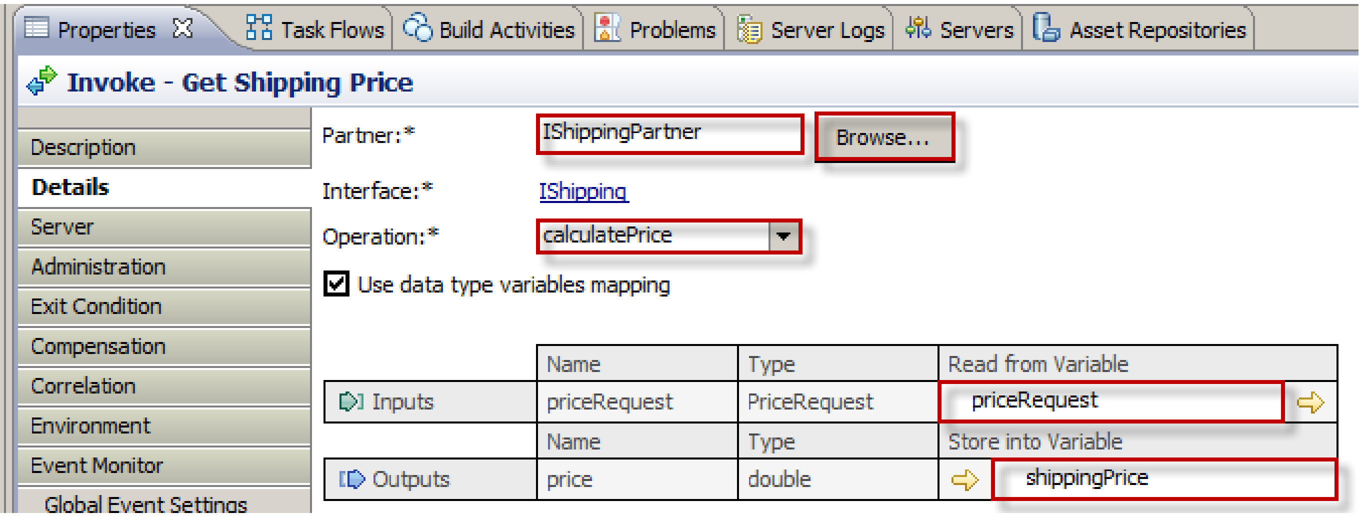
Operation:* calculatePrice

☑ Use data type variables mapping

	Name	Type	Read from Variable
Inputs	priceRequest	PriceRequest	priceRequest
	Name	Type	Store into Variable
Outputs	price	double	price

__i. Map **priceRequest** to variable **priceRequest** by clicking the **(none)** link and selecting **priceRequest** from the pop-up.
__j. Map **price** to variable **price** by clicking the **(none)** link and selecting **price** from the pop-up

__7. Add an **Invoke** activity
  - __a. From the Basic Actions palette, drag another **Invoke** activity and add it under **Get Product Price** activity
  - __b. In the properties change the **Name** from Invoke to **GetShippingPrice**
  - __c. Change the **Display name** to **Get Shipping Price.**
  - __d. Click the **Details** tab; click the **Browse...** button
  - __e. Select the **IShippingPartner** from the pop-up; click **OK**
  - __f. Map **priceRequest** to variable **priceRequest** by clicking the **(none)** link and selecting **priceRequest** from the pop-up.
  - __g. Map **price** to variable **price** by clicking the **(none)** link and selecting **price** from the pop-up.



Invoke - Get Shipping Price

Partner:* IShippingPartner

Interface:* IShipping

Operation:* calculatePrice

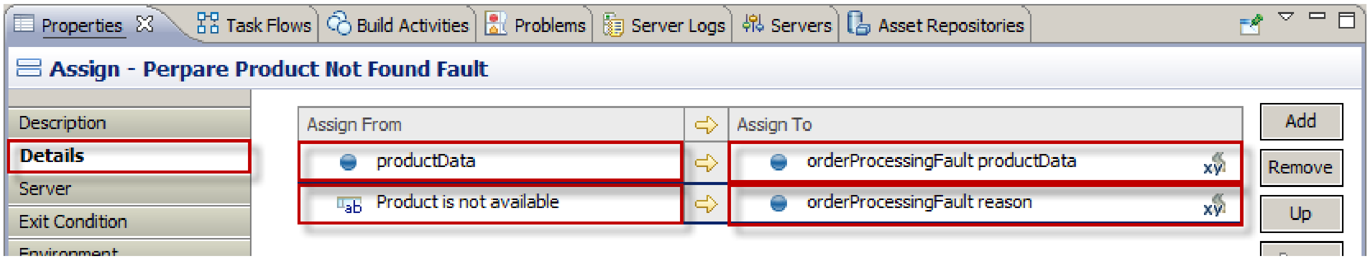
☑ Use data type variables mapping

	Name	Type	Read from Variable
Inputs	priceRequest	PriceRequest	priceRequest
	Name	Type	Store into Variable
Outputs	price	double	shippingPrice

Now to finish up the Otherwise flow.

__8. Add an **Assign** activity
  - __a. From the Basic Actions palette, drag an Assign activity Assign and add it under the **Otherwise** element.

__b. In the Properties, change the **Name** from **Assign** to **PrepareProductNotFoundFault**
__c. Change the **Display name** to **Prepare Product Not Found Fault**
__d. Click the **Details** tab
  __i. Map productData to orderProcessingFault.productData
  __ii. Add a string “Product is not available” and map it to orderProcessingFault.reason

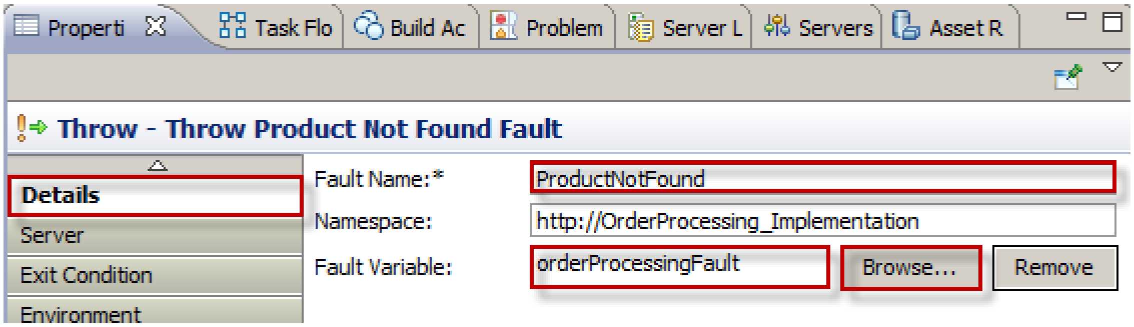


Assign - Perpare Product Not Found Fault

Assign From		Assign To
productData	⇨	orderProcessingFault productData
Product is not available	⇨	orderProcessingFault reason

__9. Add a **Throw** activity

__a. From the Faults palette, add a Throw activity Throw under the **Prepare Product Not Found Fault** activity
__b. In the Properties, change the **Name** from **Throw** to **ThrowProductNotFoundFault** and change the **Display name** to **Throw Product Not Found Fault.**
__c. Click the **Details** tab and change the **Fault Name:** to **ProductNotFound**.
  __i. Click the **Browse...** button and select **orderProcessingFault** from the pop-up

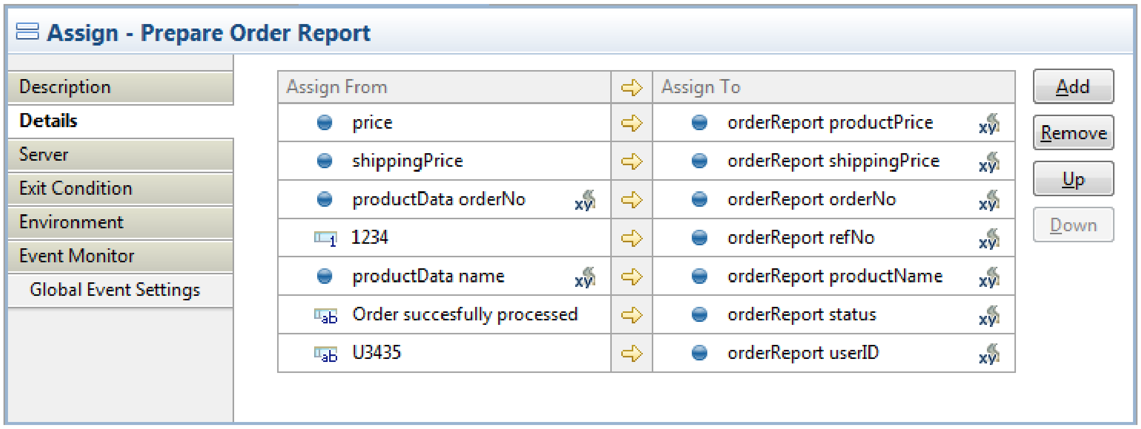


Throw - Throw Product Not Found Fault

Fault Name:* ProductNotFound
Namespace: http://OrderProcessing_Implementation
Fault Variable: orderProcessingFault

__10. Add an **Assign** activity

__a. From the Basic Actions palette, add an Assign activity Assign between the **Choice** box **Is Product Available** and the **Reply** node.
__b. In the Properties, change the **Name** from **Assign** to **PrepareOrderReport**
__c. Change the **Display name** to **Prepare Order Report**
__d. Click the **Details** tab and map the follow variables.
  __i. Map productPrice to orderReport.productPrice
  __ii. Map shippingPrice to orderReport.shippingPrice
  __iii. Map productData.orderNo to orderReport.orderNo
  __iv. Add a string “1234” and map it to orderReport.refNo
  __v. Map productData.name to orderReport.productName
  __vi. Add a string “Order successfully processed” and map it to orderReport.status
  __vii. Add a string “U3435” and map it to orderReport.userID



Assign - Prepare Order Report

Assign From		Assign To
price	⇨	orderReport productPrice
shippingPrice	⇨	orderReport shippingPrice
productData orderNo	⇨	orderReport orderNo
1234	⇨	orderReport refNo
productData name	⇨	orderReport productName
Order succesfully processed	⇨	orderReport status
U3435	⇨	orderReport userID

__11. Add an **Invoke** activity

__a. From the Basic Actions palette, add an **Invoke** activity Invoke between **Prepare Order Report** and the **Reply** node.

__b. In the Properties, change the **name** from **Invoke** to **SaveOrder**

__c. Change the **Display name** to **Save Order**.

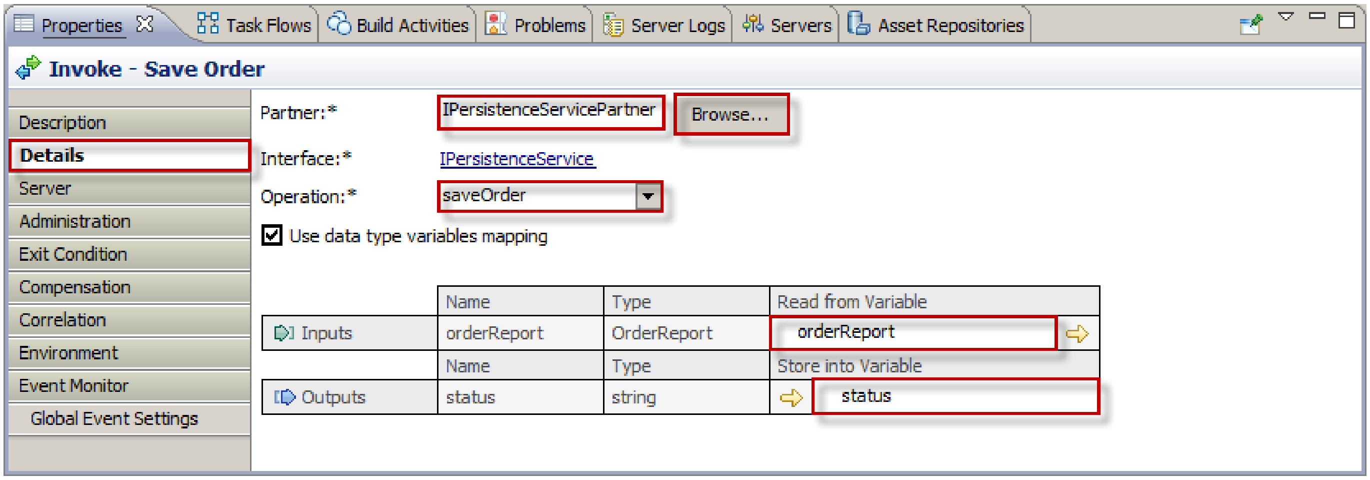
__d. Click the **Details** tab; click the **Browse...** button

__e. Select the **IPersistenceServicePartner** from the pop-up

__f. Click **OK** button.

__g. Map **orderReport** to variable **orderReport** by clicking the link **(none)** and selecting **orderReport** from the pop-up.

__h. Map **status** to variable **status** by clicking the **(none)** link and selecting **status** from the pop-up.



Invoke - Save Order

Partner:* IPersistenceServicePartner

Interface:* IPersistenceService

Operation:* saveOrder

☑ Use data type variables mapping

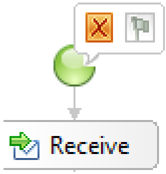
	Name	Type	Read from Variable
Inputs	orderReport	OrderReport	orderReport
	Name	Type	Store into Variable
Outputs	status	string	status

1.4.4 Add a Fault Handler to the BPEL Process

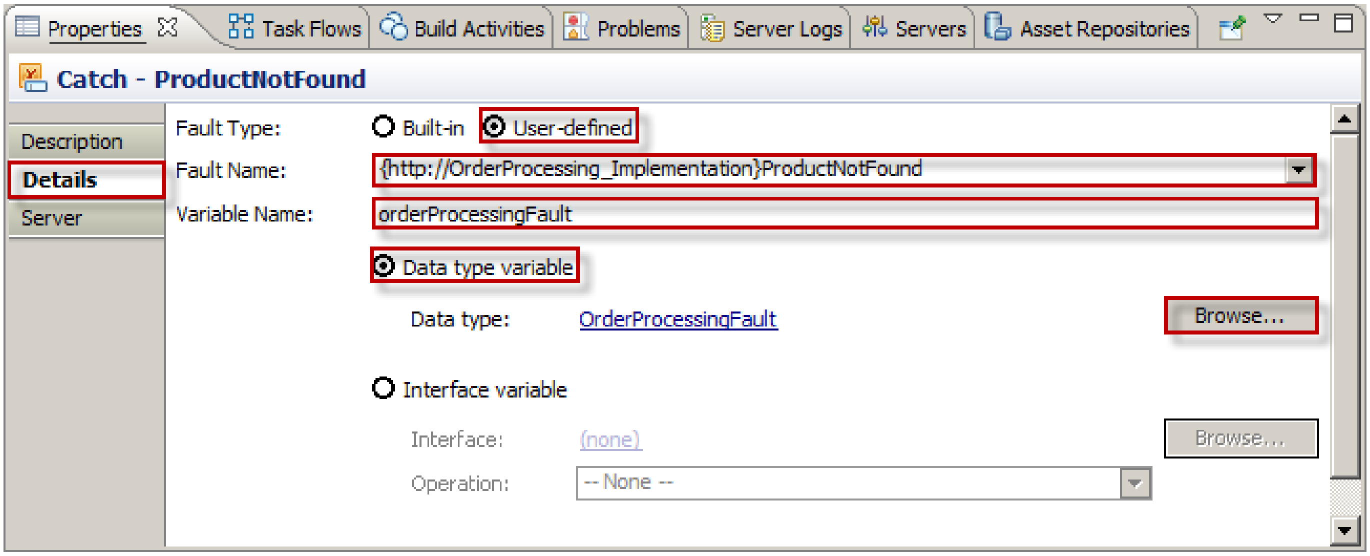
__1. Add a **Fault Handler**

__a. Highlight the green start circle, right-mouse click and choose **Add a Fault Handler** OR Select the green circle at the top of the BPEL process. A set of icons will appear.

__i. Select the **Add a Fault Handler** icon .



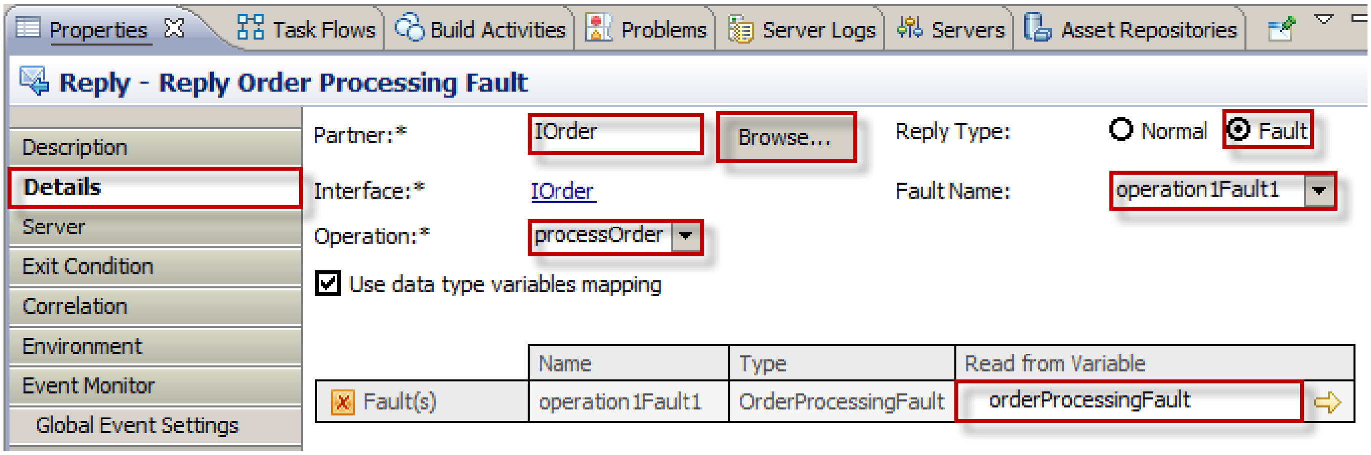
__b. In properties change the Display name from **Catch** to **Product Not Found**
__c. Click the **Details** tab
  __i. Change the **Fault Type** to **User-defined**
  __**ii.** Change the **Fault Name:** by clicking the drop down and selecting **{**http://OrderProcessing_Implementation**}ProductNotFound**
  __iii. In the **Variable Name** field, enter **orderProcessingFault**
  __iv. Click the **Data type variable** radio button
  __v. Click the **Browse...** button and select **OrderProcessingFault** from the pop-up



Catch - ProductNotFound

Fault Type: ○ Built-in ● User-defined
Fault Name: {http://OrderProcessing_Implementation}ProductNotFound
Variable Name: orderProcessingFault
● Data type variable
Data type: OrderProcessingFault
○ Interface variable
Interface: (none)
Operation: -- None --

__2. Add a **Reply Action**
  __a. From the Basic Actions palette, add a **Reply** activity Reply under the **Product Not Found** Catch activity
  __b. In the properties
  __c. Change the **Name** from **Reply** to **ReplyOrderProcessingFault**
  __d. Change the **Display name** to **Reply Order Processing Fault**.
  __e. In the Properties, click the **Details** tab
  __f. Click the **Browse...** button and select the **IOrder** partner from the pop-up menu.
  __g. Click the **OK** Button
  __h. Change the **Reply Type** to **Fault**
  __i. Map the fault to **orderProcessingFault** by clicking the **(none)** link and selecting **orderProcessingFault** from the pop-up



Reply - Reply Order Processing Fault

Partner:* IOrder

Interface:* IOrder

Operation:* processOrder

☑ Use data type variables mapping

Reply Type: ○ Normal ● Fault

Fault Name: operation1Fault1

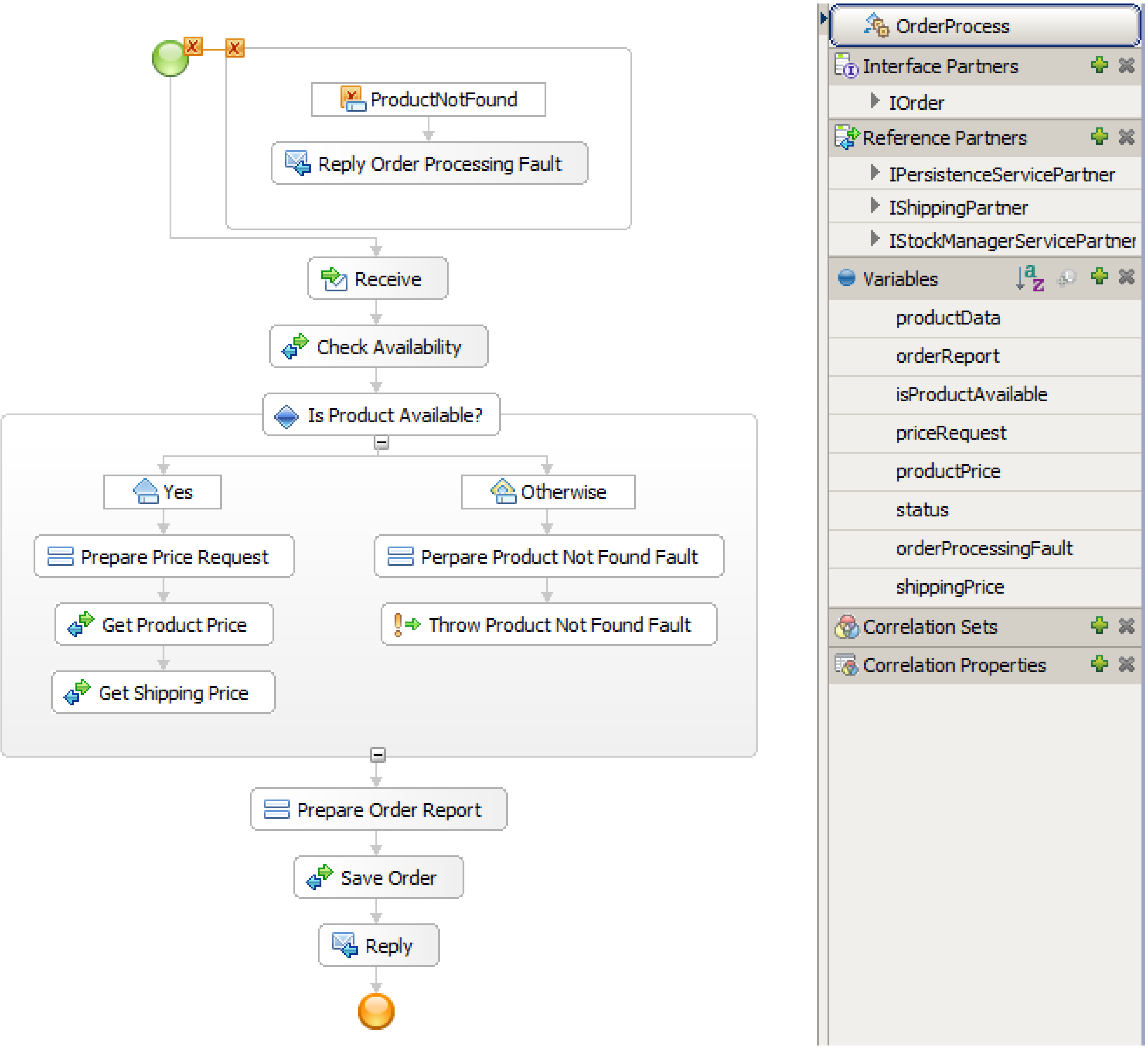
	Name	Type	Read from Variable
Fault(s)	operation1Fault1	OrderProcessingFault	orderProcessingFault

__3. Click the **Save** button or press **Ctrl+s** to save your work.

__4. Click the Problems tab and make sure there are no errors.

  __a. Fix all errors before you close the BPEL editor.

1.4.5 Completed BPEL Process – Order Processing

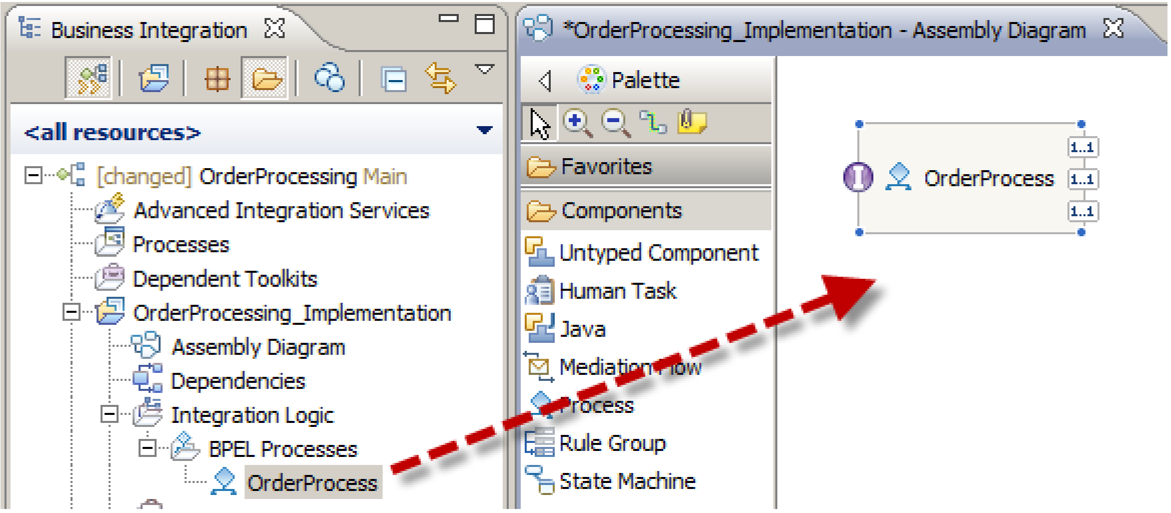


1.5 Assemble the Order Processing Module

To finish the implementation of the process, you have to create the module assembly.

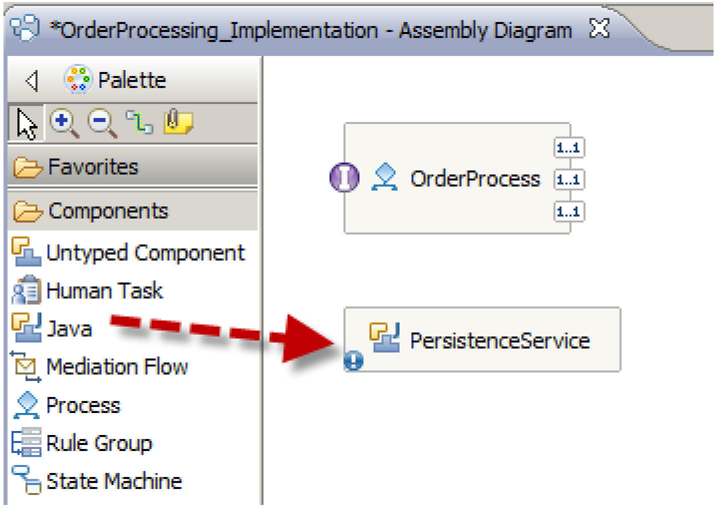
1.5.1 Add the Order Processing Process

__1. Expand the **OrderProcessing_Implementation**,
  __a. Double-click on the Assembly Diagram
__2. Expand the **Integration Logic** folder
  __a. Expand the **BPEL Processes** folder and select the **OrderProcessing** component.
__3. Drag the **OrderProcessing** BPEL component onto the Assembly Diagram canvas

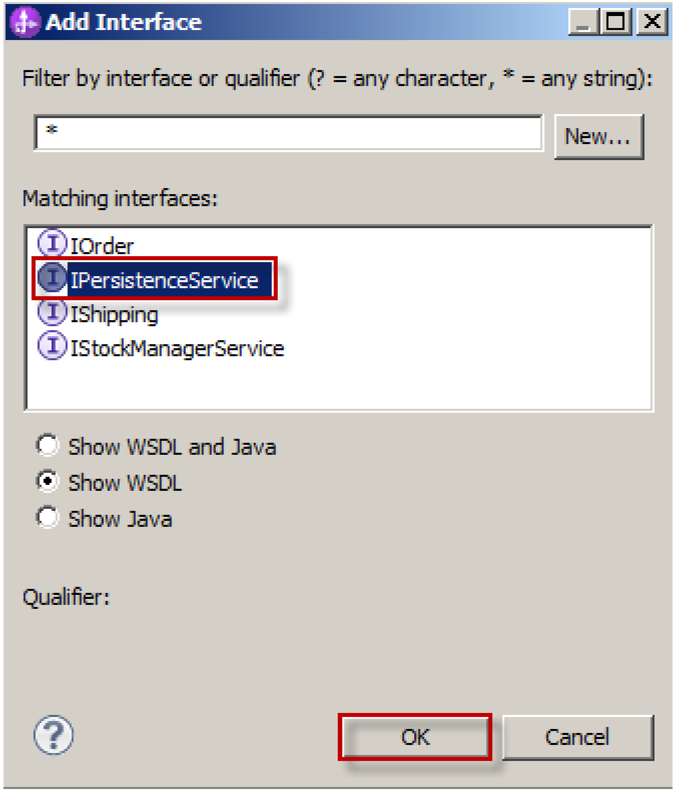


1.5.2 Add Java Components

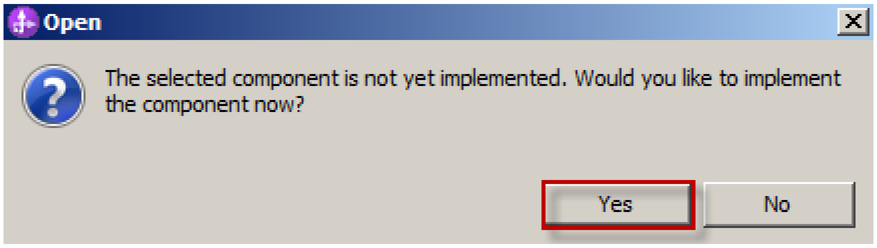
__1. Add the **IPersistenceService** Java Component
  __a. From the Components palette, drag a Java Component Java onto the Assembly Diagram canvas
  __b. In Properties change **Name** to **PersistenceService**.



__c. Click **Details**, highlight **Interfaces**, right-click and select **Add Interface** from pop-up menu.
  __i. Select the **IPersistenceService** interface from the list.
  __ii. Click **OK**

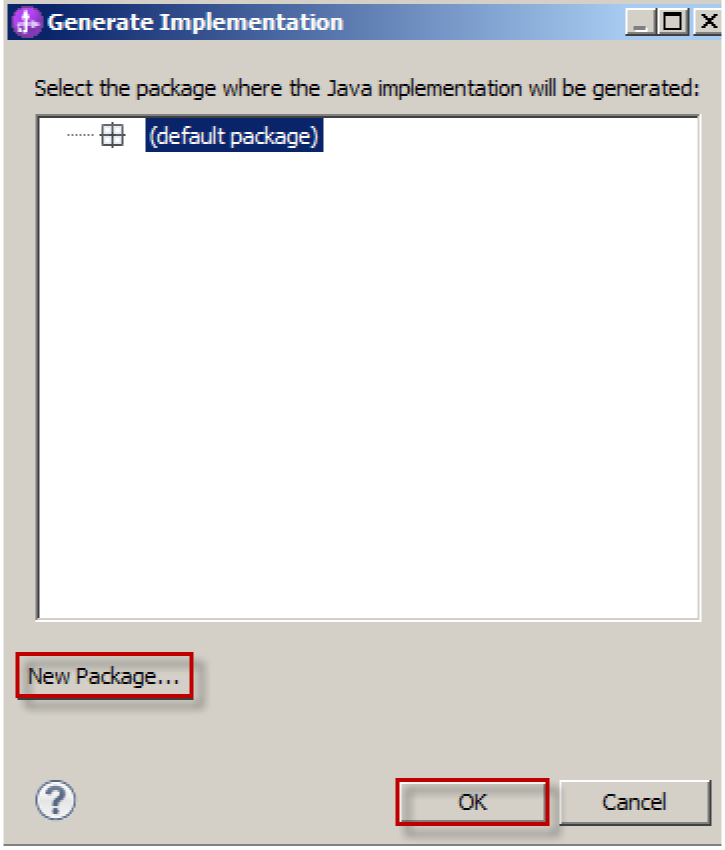


__d. Double-click on the new Java component.
__e. When asked about the implementation, click the **Yes** button



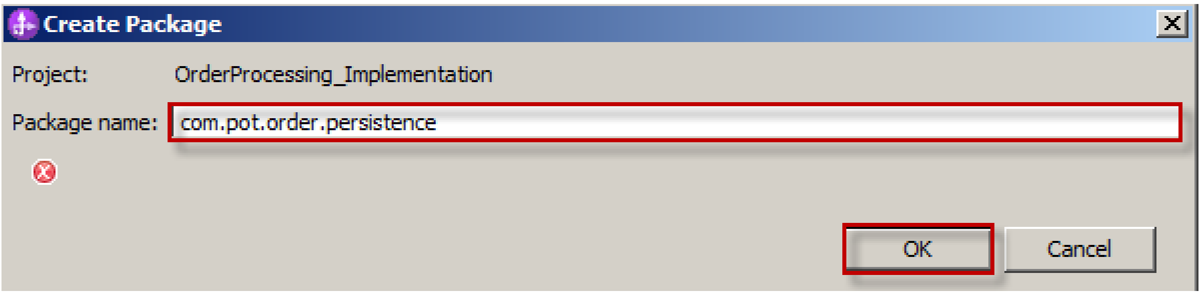
The selected component is not yet implemented. Would you like to implement the component now?

__f. Click **New Package...** button



Select the package where the Java implementation will be generated:

__g. Enter **com.pot.order.persistence** in the **Package name** field.
__h. Click the **OK** button.



Project: OrderProcessing_Implementation

Package name: com.pot.order.persistence

__i. Click the **OK** button again and the Java Editior will appear showing the generated Java implementation __j. Modify the **saveOrder()** method to look like this

  __i. The code changes are also available in Persistence.txt

```
      public String saveOrder(DataObject productData) {
            // To get or set attributes for DataObject productData, use the APIs
as shown below:
            // To set a string attribute in productData, use
productData.setString(stringAttributeName, stringValue)
            // To get a string attribute in productData, use
productData.getString(stringAttributeName)
            // To set a dataObject attribute in productData, use
productData.setDataObject(stringAttributeName, dataObjectValue)
            // To get a dataObject attribute in productData, use
productData.getDataObject(stringAttributeName)
            System.out.println("saveOrder..........");
            return "Successfully saved";
      }
```

__k. Modify the **undoSaveOrder()** method to look like this:

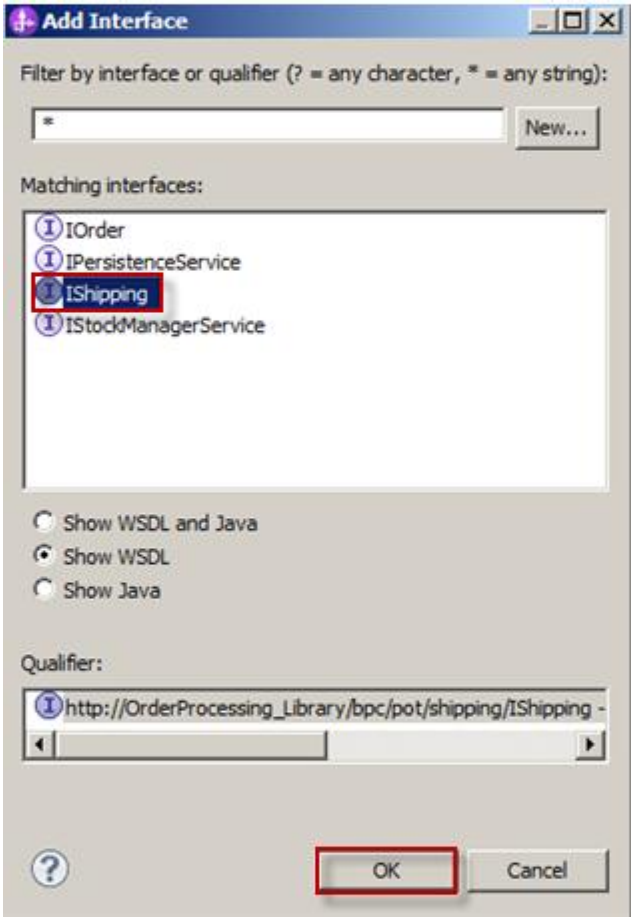
__l. The code changes are also available in Persistence.txt

```
      public String undoSaveOrder(DataObject productData) {
            // To get or set attributes for DataObject productData, use the APIs
as shown below:
            // To set a string attribute in productData, use
productData.setString(stringAttributeName, stringValue)
            // To get a string attribute in productData, use
productData.getString(stringAttributeName)
            // To set a dataObject attribute in productData, use
productData.setDataObject(stringAttributeName, dataObjectValue)
            // To get a dataObject attribute in productData, use
productData.getDataObject(stringAttributeName)

            System.out.println("undoSaveOrder..........");
            return "Successfully unsaved";

       }
```

__m. Close the Java editor and save the changes.

__2. Similarly, add the **IShippingService** Java Component

__a. Drag a Java Component Java onto the Assembly Diagram.
__b. In properties change **Name** to **ShippingService**.
__c. Click **Details**; highlight **Interfaces** right-click and select **Add Interface** from pop-up menu.
__d. Select the **IShipping** interface from the list.



Add Interface

Filter by interface or qualifier (? = any character, * = any string):

`*`

Matching interfaces:

- IOrder
- IPersistenceService
- IShipping
- IStockManagerService

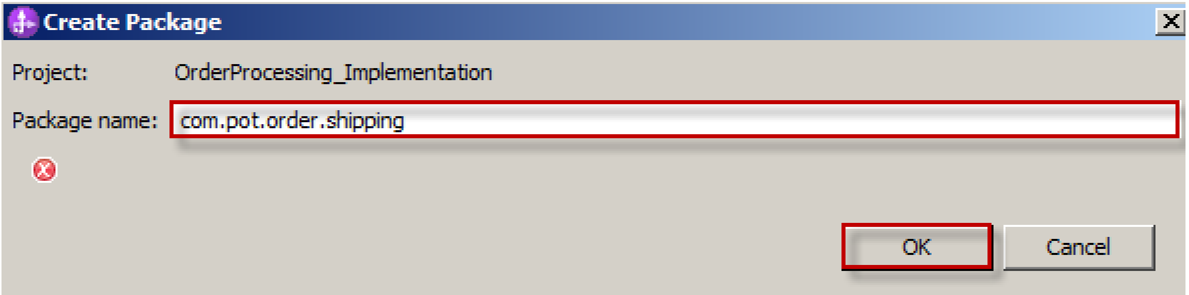
○ Show WSDL and Java
● Show WSDL
○ Show Java

Qualifier:

http://OrderProcessing_Library/bpc/pot/shipping/IShipping -

__e. Double-click on the new Java component.
__f. When asked about the implementation, click the **Yes** button

__g. Create a **New Package...** and name it **com.pot.order.shipping**



Create Package

Project: OrderProcessing_Implementation

Package name: com.pot.order.shipping

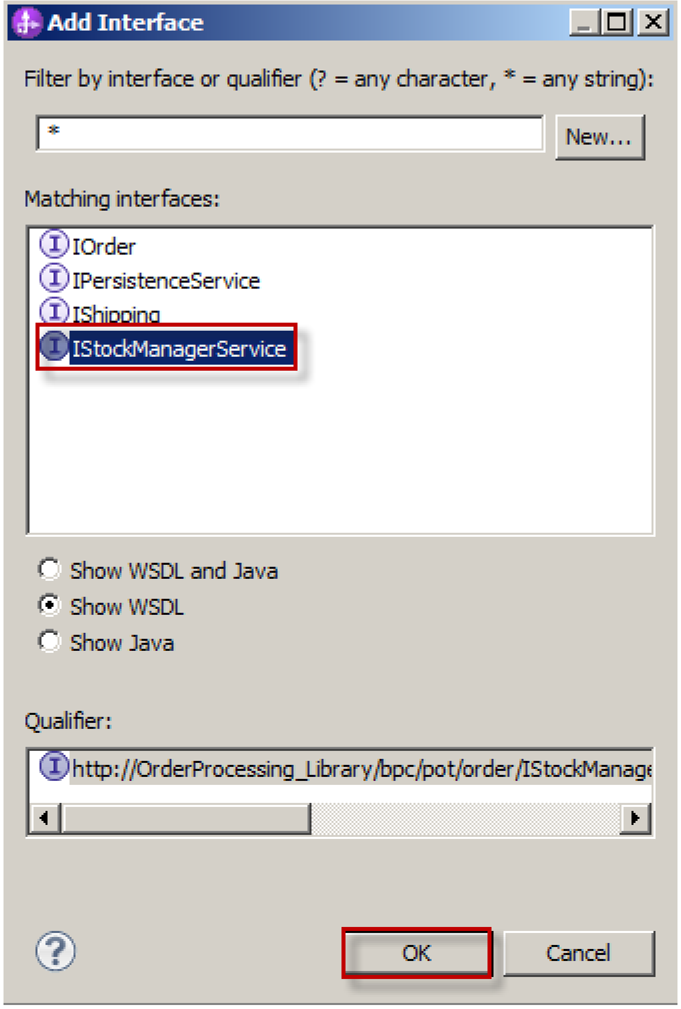
__h. In the Java implementation, modify method **calculatePrice()** method to look like this:

```
      public Double calculatePrice(DataObject priceRequest) {
            // To get or set attributes for DataObject priceRequest, use the APIs
as shown below:
            // To set a string attribute in priceRequest, use
priceRequest.setString(stringAttributeName, stringValue)
            // To get a string attribute in priceRequest, use
priceRequest.getString(stringAttributeName)
            // To set a dataObject attribute in priceRequest, use
priceRequest.setDataObject(stringAttributeName, dataObjectValue)
            // To get a dataObject attribute in priceRequest, use
priceRequest.getDataObject(stringAttributeName)
            double price = 2.0;
            if(priceRequest != null)
            {
                  String size = priceRequest.getString("size");

                  if("S".equals(size))price = 2.0;
                  else if("M".equals(size))price = 4.0;
                  else if("L".equals(size))price = 6.0;
                  else if("XL".equals(size))price = 8.0;
            }

            //initialize the Output variable:

            return price;

      }
```

__i. The code changes are also available in ShippingService.txt
__j. Close the Java editor and save the changes.

__3. Similarly, add the **IStockManagerService** Java Component

- __a. From the palette, drag a Java Component Java onto the Assembly Diagram.
- __b. In Properties, change **Name** to **StockManagerService**.
- __c. Click **Details**; highlight **Interfaces**, right-click and select **Add Interface** from pop-up menu.
- __d. Select the **IStockManagerService** interface from the list.
- __e. Click **OK**



Filter by interface or qualifier (? = any character, * = any string):

*

Matching interfaces:

- IOrder
- IPersistenceService
- IShipping
- IStockManagerService

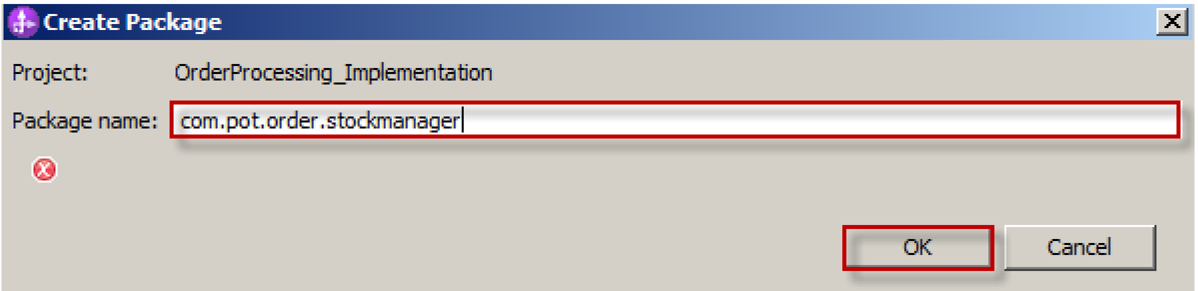
○ Show WSDL and Java
● Show WSDL
○ Show Java

Qualifier:

http://OrderProcessing_Library/bpc/pot/order/IStockManage

- __f. Double-click on the new Java component.
- __g. When asked about the implementation, click the **Yes** button

- __h. Create a **New Package...** and name it **com.pot.order.stockmanager**


Create Package

Project: OrderProcessing_Implementation

Package name: com.pot.order.stockmanager

__i. In the Java implementation modify the **checkAvailability()** method to look like this:

```
      public Boolean checkAvailability(DataObject productData) {
            // To get or set attributes for DataObject productData, use the APIs
as shown below:
            // To set a string attribute in productData, use
productData.setString(stringAttributeName, stringValue)
            // To get a string attribute in productData, use
productData.getString(stringAttributeName)
            // To set a dataObject attribute in productData, use
productData.setDataObject(stringAttributeName, dataObjectValue)
            // To get a dataObject attribute in productData, use
productData.getDataObject(stringAttributeName)
            System.out.println("checkAvailability(DataObject) - ENTRY.");

            String kind = "";
            int number = 0;

            if (productData != null) {
                  System.out.println("productData != null");
                  kind = productData.getString("kind");
                  number = productData.getInt("number");
                  System.out.println("requested number of kind " + kind + ": "
                                    + number);
            }

            if (number > 0) {
              if (!"".equals(kind)) {
                  System.out.println("checkAvailability(DataObject) - EXIT.
true");
                  return Boolean.TRUE;
              }
            }
                  System.out.println("checkAvailability(DataObject) - EXIT.
false");
                  return Boolean.FALSE;

      }
```

__j. The code changes are also available in StockManagerService.txt

__k. Modify the **calculatePrice()** method to look like this:

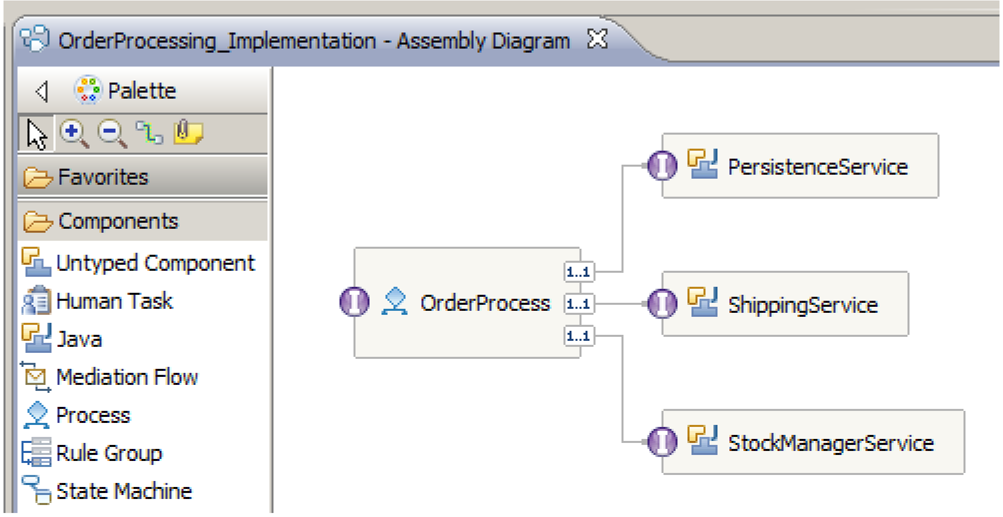
```
public Double calculatePrice(DataObject priceRequest) {
            // To get or set attributes for DataObject productData, use the APIs
as shown below:
            // To set a string attribute in productData, use
productData.setString(stringAttributeName, stringValue)
            // To get a string attribute in productData, use
productData.getString(stringAttributeName)
            // To set a dataObject attribute in productData, use
productData.setDataObject(stringAttributeName, dataObjectValue)
            // To get a dataObject attribute in productData, use
productData.getDataObject(stringAttributeName)
            double price = 2.0;
            int number = 1;
            if(priceRequest != null)
            {
                  String size = priceRequest.getString("size");
                  number = priceRequest.getInt("number");

                  if("S".equals(size))price = 2.0;
                  else if("M".equals(size))price = 4.0;
                  else if("L".equals(size))price = 6.0;
                  else if("XL".equals(size))price = 8.0;
            }

            //initialize the Output variable:

            return price*number;

      }
```

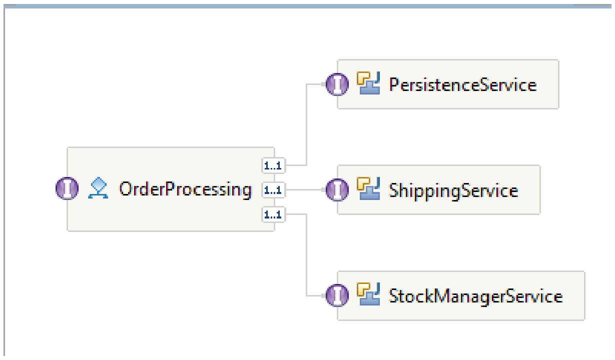
__l. The code changes are also available in StockManagerService.txt.
__m. Close the Java editor and save the changes.

__n. You may want to save the changes to the Assembly Diagram.

1.5.3 Wire the Assembly Diagram

__1. Click on the Wire icon.
   __a. Hover over the interface in the OrderProcess component. You will see which Java component to wire to.
__2. Connect a wire from the **PersistenceService** reference of the **OrderProcessing** component to the interface of the **PersistenceService** component.
__3. Similarly connect a wire from the **ShippingService** reference of the **OrderProcessing** component to the interface of the **ShippingService** component
__4. And connect a wire from the **StockManagerService** reference of the **OrderProcessing** component to the interface of the **StockManagerService** component.
__5. Click **Save** or press **Ctrl+s** to save the edits. The resulting Assembly Diagram should look as follows.

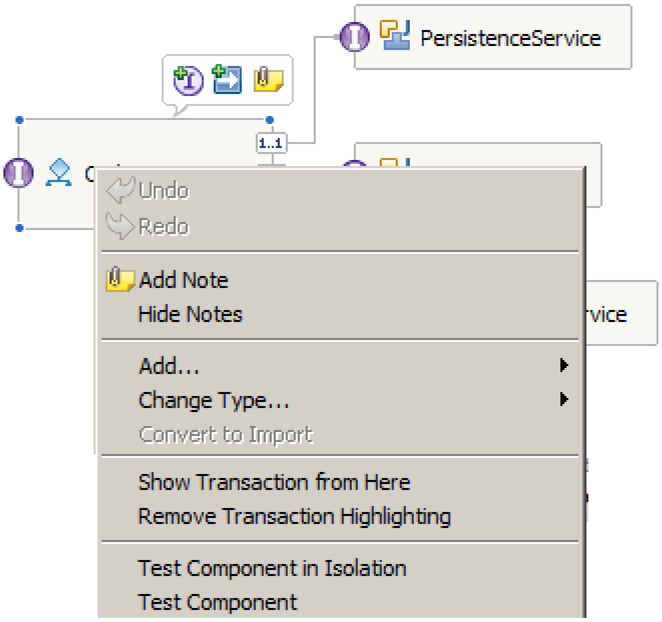




1.6 Testing the Process

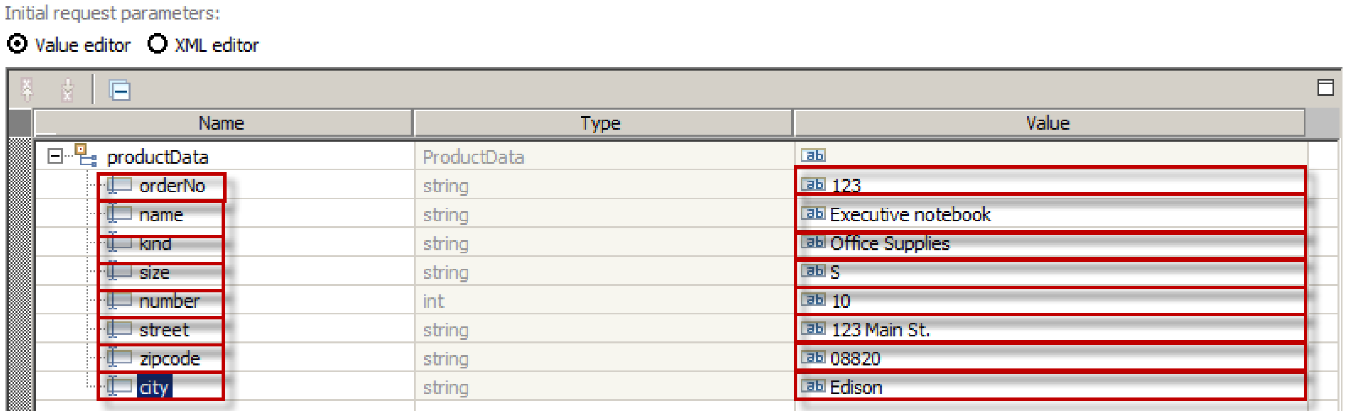
1.6.1 Positive Results Test

__1. In the Assembly Diagram editor, right-click on the **OrderProcessing** component and select **Test Component**



__2. Within the test client, fill in the Initial Request Parameters

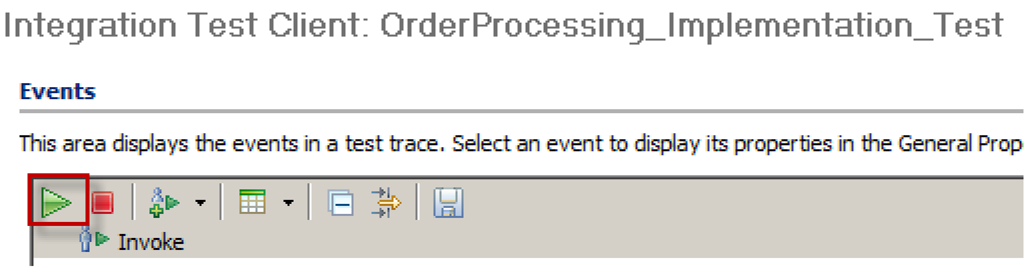
- __a. Give **orderNo** a value of **123**
- __b. Give **name** a value of **Executive notebook**
- __c. Give **kind** a value of **Office Supplies**
- __d. Give **size** a value of **S**
- __e. Give **number** a value of **10**
- __f. Give **street** a value of **123 Main St.**
- __g. Give **zipcode** a value of **08820**
- __h. Give **city** a value of **Edison**


Initial request parameters:

● Value editor ○ XML editor

Name	Type	Value
productData	ProductData	
orderNo	string	123
name	string	Executive notebook
kind	string	Office Supplies
size	string	S
number	int	10
street	string	123 Main St.
zipcode	string	08820
city	string	Edison

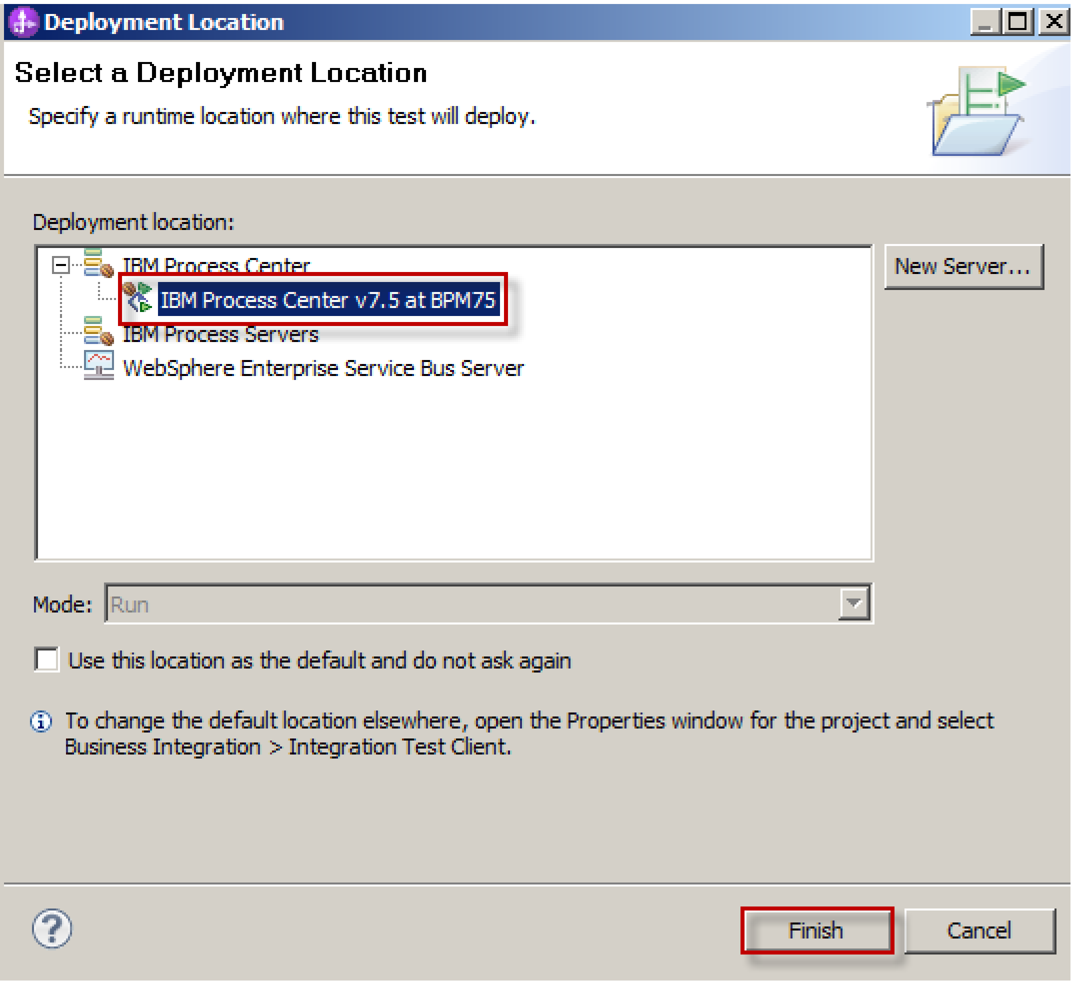
__3. Run the test by clicking the **Continue** (green arrow) button


Integration Test Client: OrderProcessing_Implementation_Test

Events

This area displays the events in a test trace. Select an event to display its properties in the General Prop

__5. A window will pop-up and ask which run-time environment to use.
  __a. Make sure the IBM Process Center is selected
  __b. Click the **Finish** button


Deployment Location

Select a Deployment Location

Specify a runtime location where this test will deploy.

Deployment location:

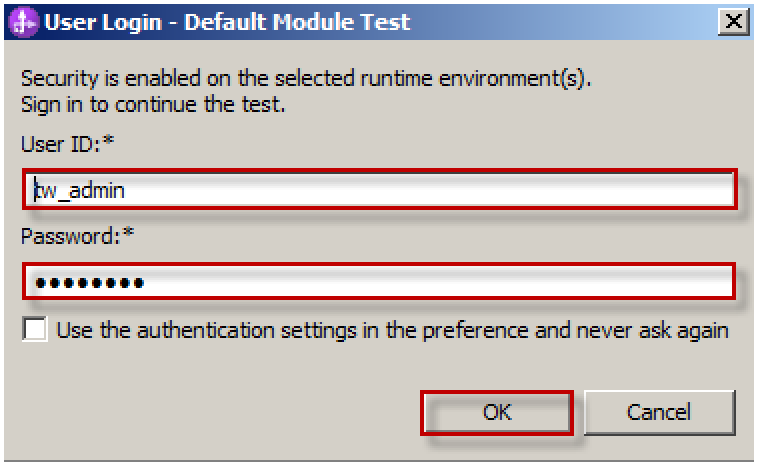
- IBM Process Center
  - IBM Process Center v7.5 at BPM75
- IBM Process Servers
- WebSphere Enterprise Service Bus Server

Mode: Run

☐ Use this location as the default and do not ask again

To change the default location elsewhere, open the Properties window for the project and select Business Integration > Integration Test Client.

__6. Provide the login credentials and click OK.



User Login - Default Module Test

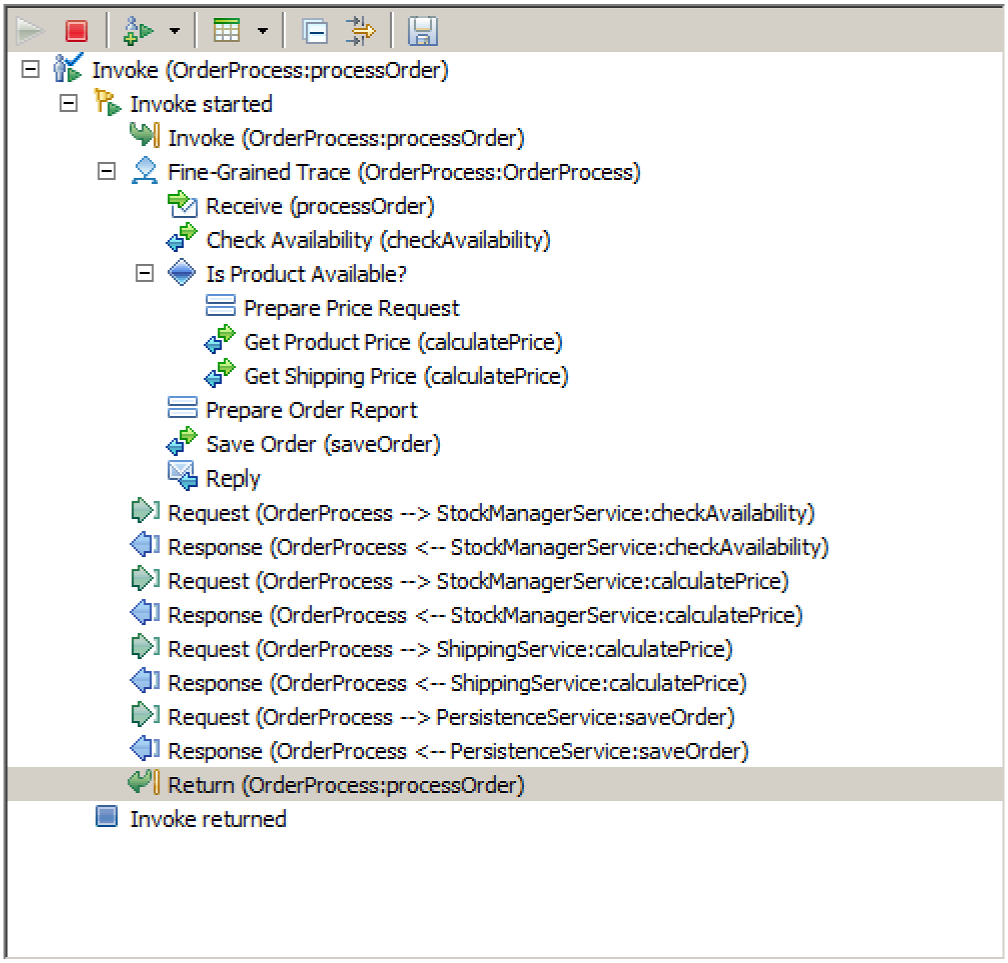
Security is enabled on the selected runtime environment(s).
Sign in to continue the test.

User ID:* tw_admin

Password:* ●●●●●●●●●

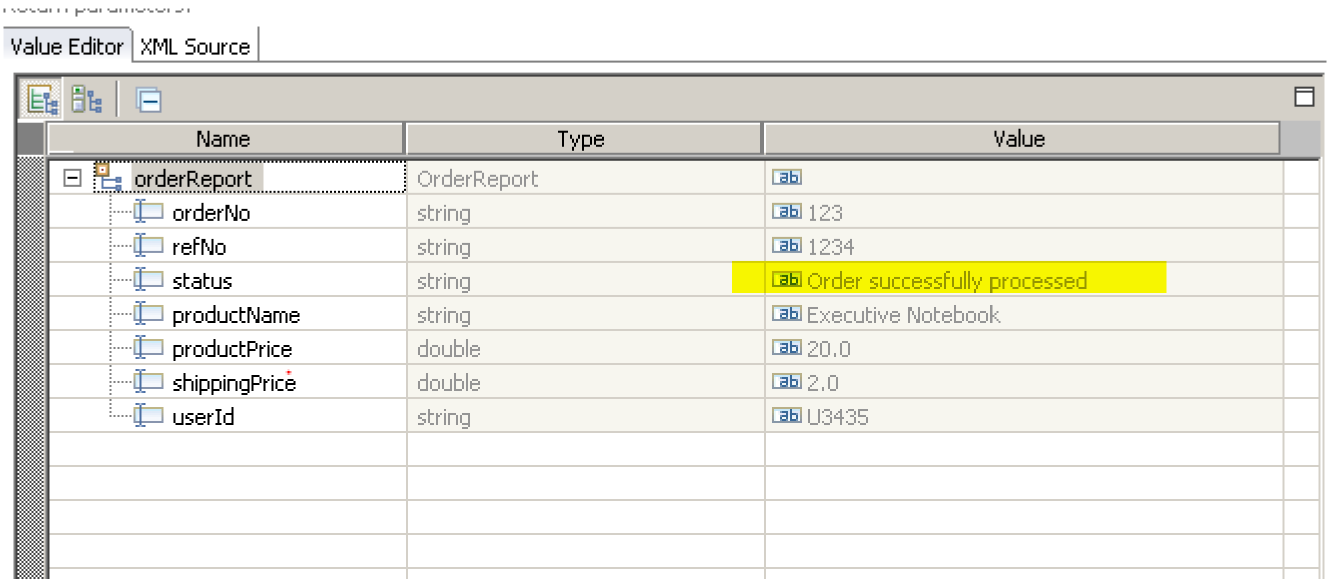
☐ Use the authentication settings in the preference and never ask again

__7. The process is deployed to the runtime and event trace is displayed.



```
Invoke (OrderProcess:processOrder)
  Invoke started
    Invoke (OrderProcess:processOrder)
    Fine-Grained Trace (OrderProcess:OrderProcess)
      Receive (processOrder)
      Check Availability (checkAvailability)
      Is Product Available?
        Prepare Price Request
        Get Product Price (calculatePrice)
        Get Shipping Price (calculatePrice)
      Prepare Order Report
      Save Order (saveOrder)
      Reply
    Request (OrderProcess --> StockManagerService:checkAvailability)
    Response (OrderProcess <-- StockManagerService:checkAvailability)
    Request (OrderProcess --> StockManagerService:calculatePrice)
    Response (OrderProcess <-- StockManagerService:calculatePrice)
    Request (OrderProcess --> ShippingService:calculatePrice)
    Response (OrderProcess <-- ShippingService:calculatePrice)
    Request (OrderProcess --> PersistenceService:saveOrder)
    Response (OrderProcess <-- PersistenceService:saveOrder)
    Return (OrderProcess:processOrder)
  Invoke returned
```

__8. Review the results you should have a **Order successfully processed**.

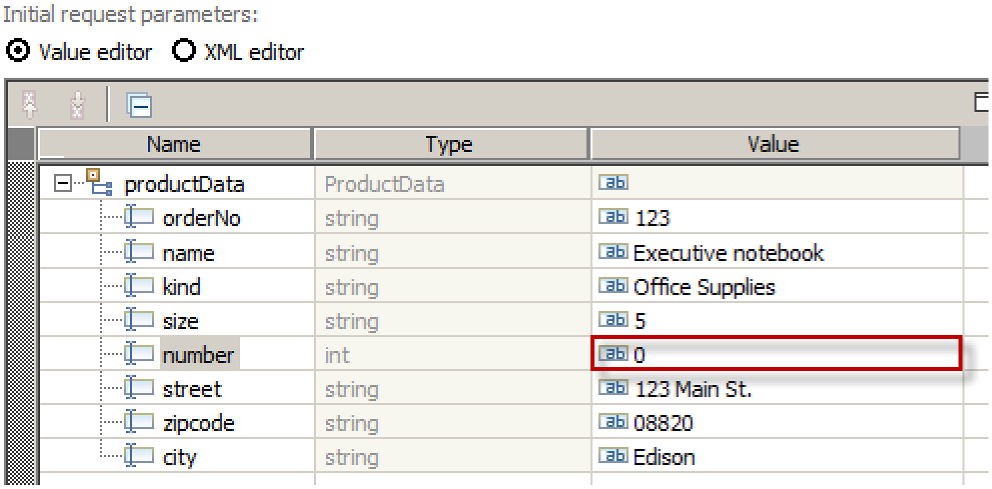


Name	Type	Value
orderReport	OrderReport	
orderNo	string	123
refNo	string	1234
status	string	Order successfully processed
productName	string	Executive Notebook
productPrice	double	20.0
shippingPrice	double	2.0
userId	string	U3435

1.6.2 Negative Results Test

After a successful run you can run a test that throws a fault.

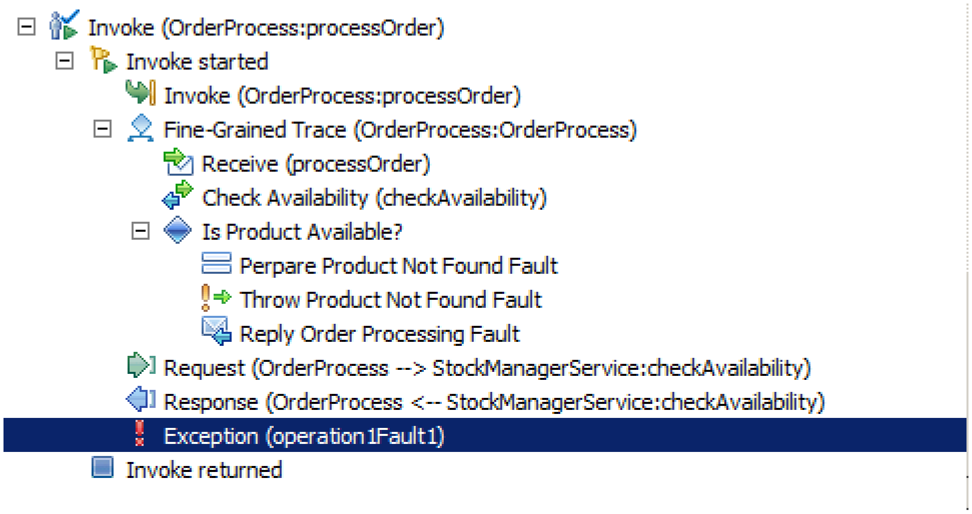
__1. Click the Invoke button to perform another run.
__2. Set the **number** to 0


Initial request parameters:

● Value editor ○ XML editor

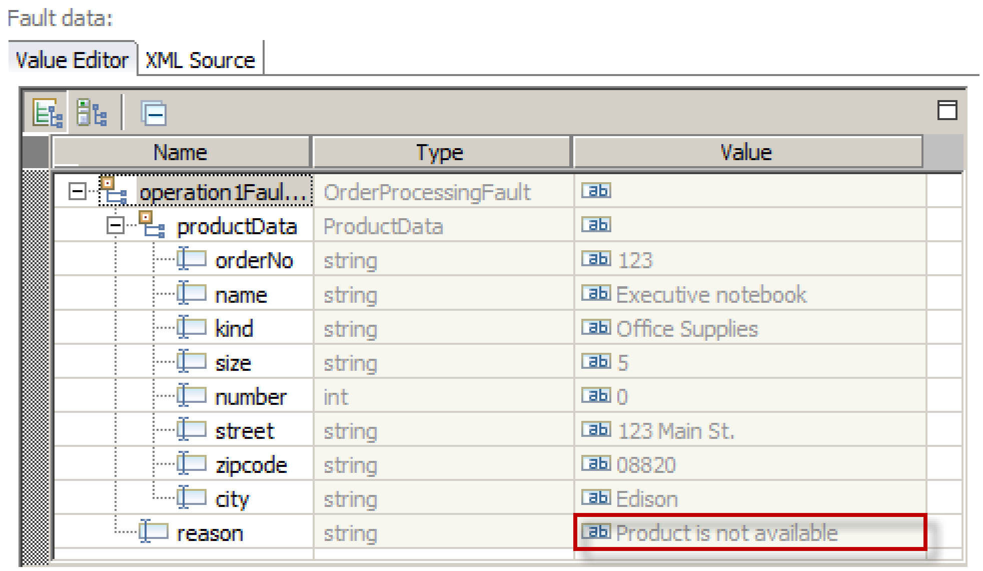
Name	Type	Value
productData	ProductData	
orderNo	string	123
name	string	Executive notebook
kind	string	Office Supplies
size	string	5
number	int	0
street	string	123 Main St.
zipcode	string	08820
city	string	Edison

__3. Click the **Continue** button



```
⊟ Invoke (OrderProcess:processOrder)
    ⊟ Invoke started
        Invoke (OrderProcess:processOrder)
        ⊟ Fine-Grained Trace (OrderProcess:OrderProcess)
            Receive (processOrder)
            Check Availability (checkAvailability)
            ⊟ Is Product Available?
                Perpare Product Not Found Fault
                Throw Product Not Found Fault
                Reply Order Processing Fault
        Request (OrderProcess --> StockManagerService:checkAvailability)
        Response (OrderProcess <-- StockManagerService:checkAvailability)
        Exception (operation1Fault1)
    Invoke returned
```

__4. If you check the output, you will see the reason – Product is not available.



Fault data:

Name	Type	Value
⊟ operation1Faul...	OrderProcessingFault	
⊟ productData	ProductData	
orderNo	string	123
name	string	Executive notebook
kind	string	Office Supplies
size	string	5
number	int	0
street	string	123 Main St.
zipcode	string	08820
city	string	Edison
reason	string	Product is not available

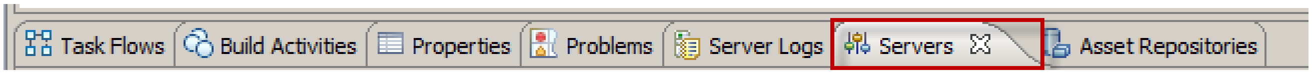
1.7 Summary

Congratulations! You have completed the BPM v7.5 STP lab. In this module we showed you how to create a BPEL business process with fault handlers. The lab used the IBM Integration Designer (IID) to develop and test the business process.

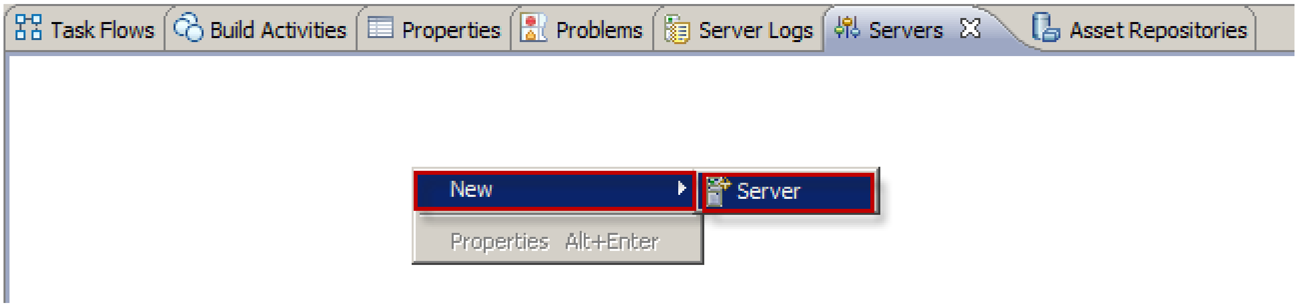
Appendix – Creating a Test Server Environment

The Process Center has a number of installation options available. In this Lab the Process Server is installed within the same JVM as the Process Center so in this step we will use the Process Center Server for testing our modules.

__1. Click the **Servers** tab

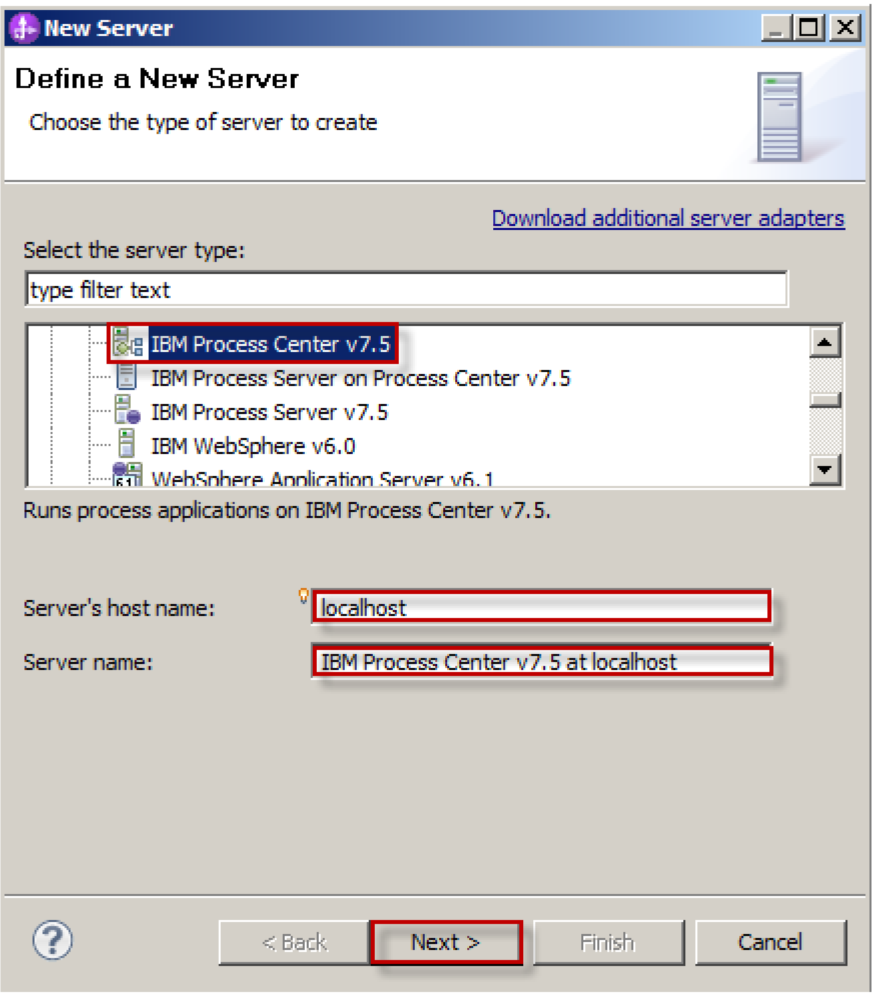


__a. Right-click in the open area in order to get the new server menu.
__b. Click **New > Server**



__2. The New Server dialog will appear.

__a. Select **IBM Process Center v7.5** from the **Select the server type:** list.
__b. Leave the **Server's host name:** default to **localhost**
__c. Leave the **Server name:** default to **IBM Process Center v7.5 at localhost**
__d. Click the **Next >** button

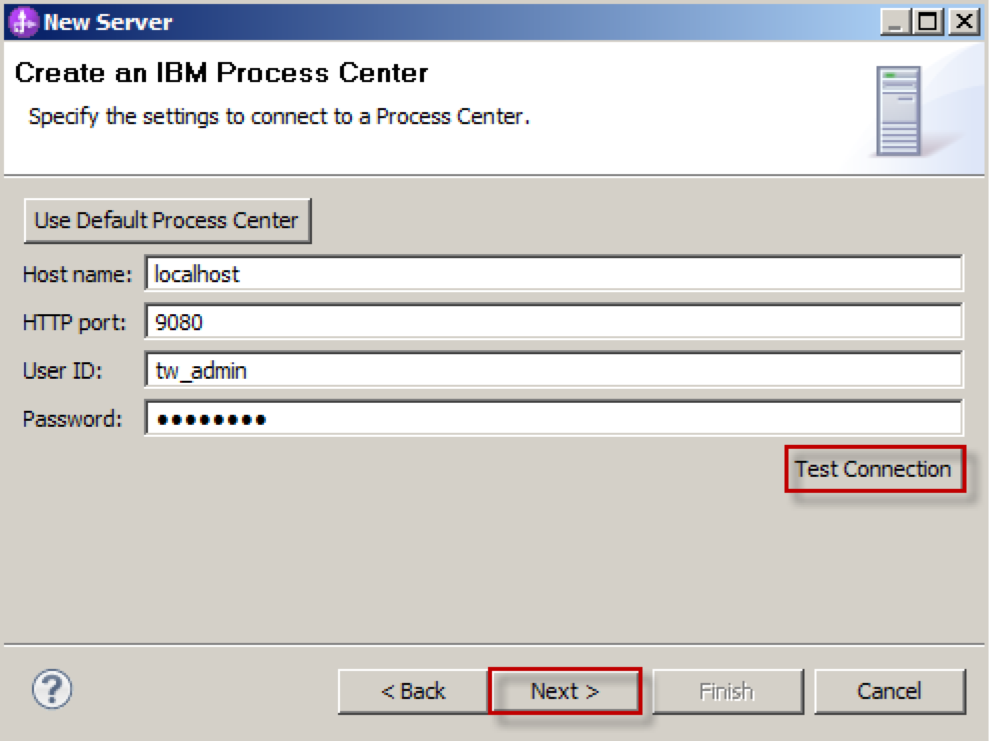


Runs process applications on IBM Process Center v7.5.

Server's host name: localhost

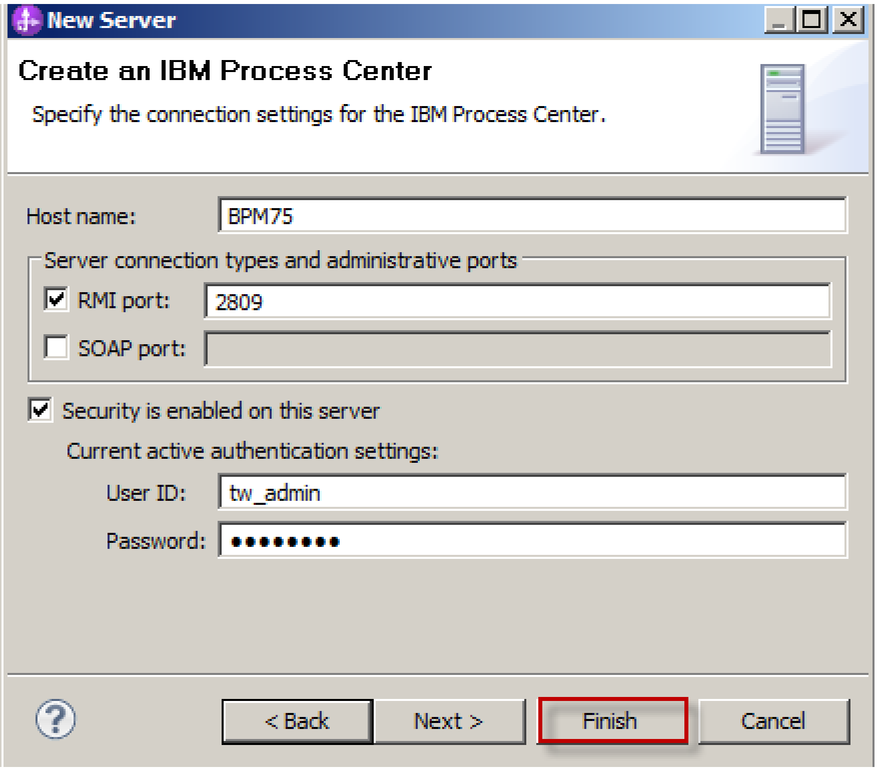
Server name: IBM Process Center v7.5 at localhost

__3. Click the **Test Connection** button. If it is unsuccessful adjust the parameters or make sure the Process Center is up and running.


New Server
Create an IBM Process Center
Specify the settings to connect to a Process Center.
Use Default Process Center
Test Connection
< Back
Next >
Finish
Cancel

Host name: localhost
HTTP port: 9080
User ID: tw_admin
Password: ●●●●●●●●●

__4. If the connection is successful, click Next and then click **Finish.**


New Server
Create an IBM Process Center
Specify the connection settings for the IBM Process Center.
Server connection types and administrative ports
Current active authentication settings:
< Back
Next >
Finish
Cancel

Host name: BPM75

☑ RMI port: 2809

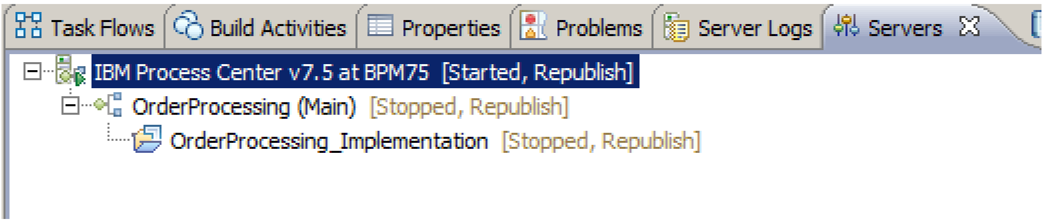
☐ SOAP port:

☑ Security is enabled on this server

User ID: tw_admin
Password: ●●●●●●●●●

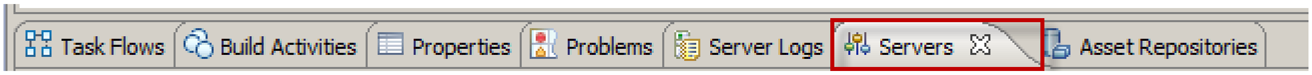
__5. You should see a Test Server that you can use to test your business process applications with.

__a. If you expand the server tree you will see that OrderProcessing Main and Implementation have been deployed to the test server.

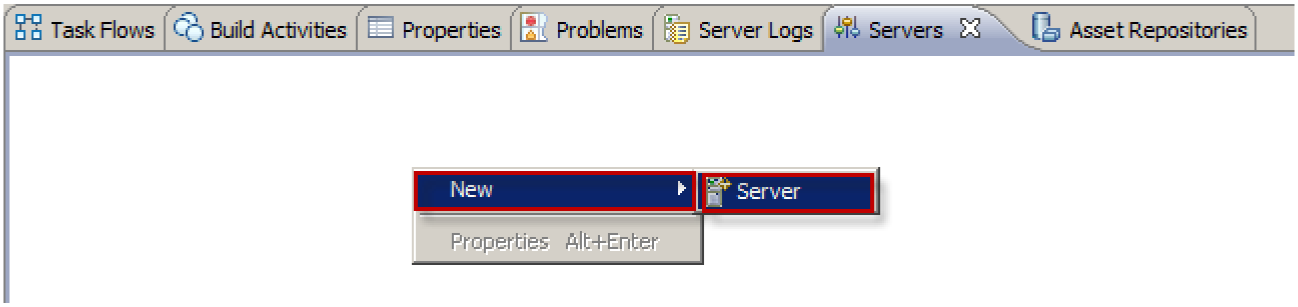

Task Flows
Build Activities
Properties
Problems
Server Logs
Servers
IBM Process Center v7.5 at BPM75 [Started, Republish]
OrderProcessing (Main) [Stopped, Republish]
OrderProcessing_Implementation [Stopped, Republish]

The Process Center has a number of installation options available. In this Lab the Process Server is installed within the same JVM as the Process Center so in this step we will use the Process Center Server for testing our modules.

__6. Click the **Servers** tab

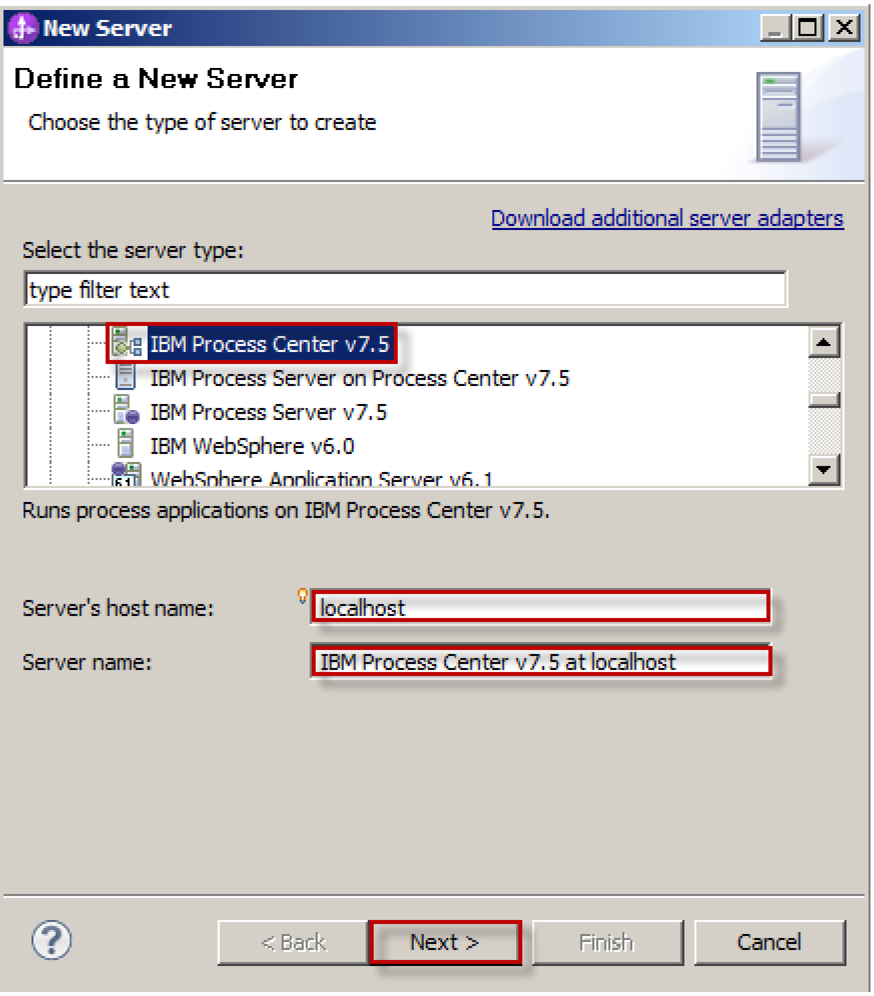


__a. Right-click in the open area in order to get the new server menu.
__b. Click **New > Server**



__7. The New Server dialog will appear.

__a. Select **IBM Process Center v7.5** from the **Select the server type:** list.
__b. Leave the **Server's host name:** default to **localhost**
__c. Leave the **Server name:** default to **IBM Process Center v7.5 at localhost**
__d. Click the **Next >** button



Define a New Server

Choose the type of server to create

Download additional server adapters

Select the server type:

type filter text

IBM Process Center v7.5
IBM Process Server on Process Center v7.5
IBM Process Server v7.5
IBM WebSphere v6.0
WebSphere Application Server v6.1

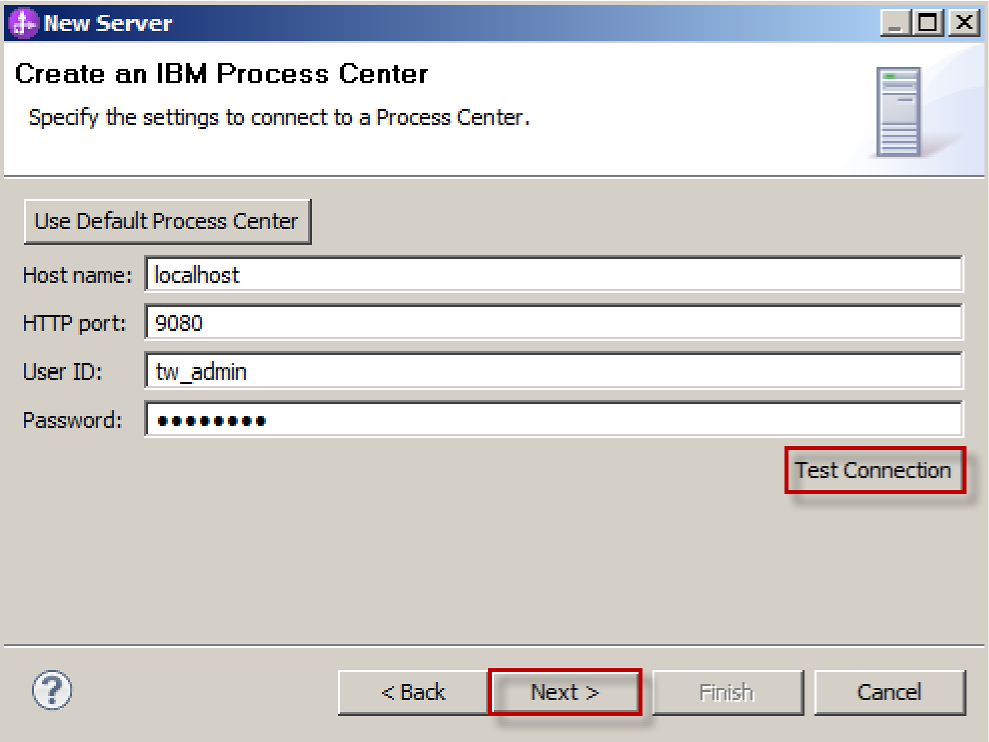
Runs process applications on IBM Process Center v7.5.

Server's host name: localhost

Server name: IBM Process Center v7.5 at localhost

< Back
Next >
Finish
Cancel

__8. Click the **Test Connection** button. If it is unsuccessful adjust the parameters or make sure the Process Center is up and running.



Create an IBM Process Center

Specify the settings to connect to a Process Center.

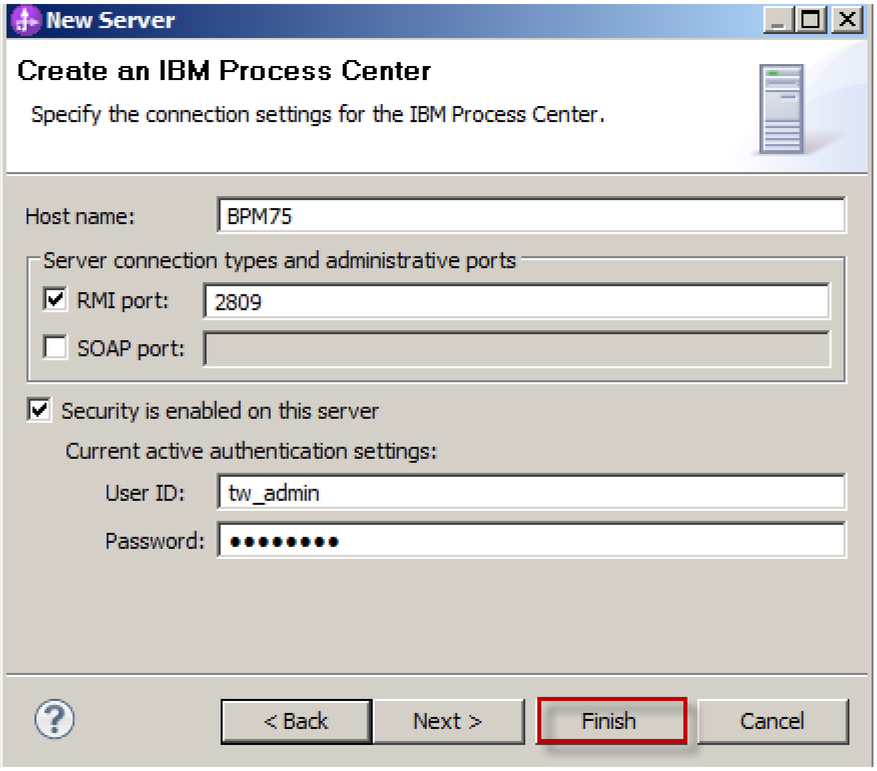
Host name: localhost

HTTP port: 9080

User ID: tw_admin

Password: •••••••••

__9. If the connection is successful, click Next and then click **Finish.**



Create an IBM Process Center

Specify the connection settings for the IBM Process Center.

Host name: BPM75

Server connection types and administrative ports

☑ RMI port: 2809

☐ SOAP port:

☑ Security is enabled on this server

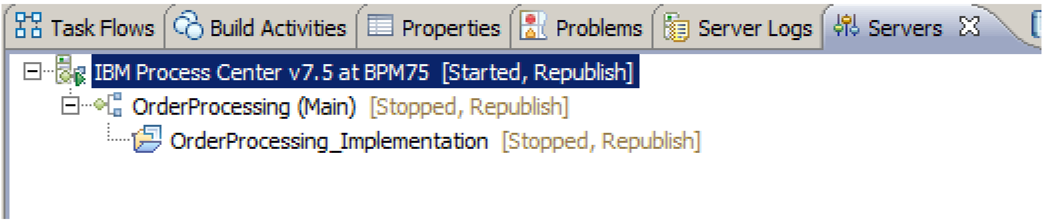
Current active authentication settings:

User ID: tw_admin

Password: •••••••••

__10. You should see a Test Server that you can use to test your business process applications with.

__a. If you expand the server tree you will see that OrderProcessing Main and Implementation have been deployed to the test server.



IBM Process Center v7.5 at BPM75 [Started, Republish]

OrderProcessing (Main) [Stopped, Republish]

OrderProcessing_Implementation [Stopped, Republish]